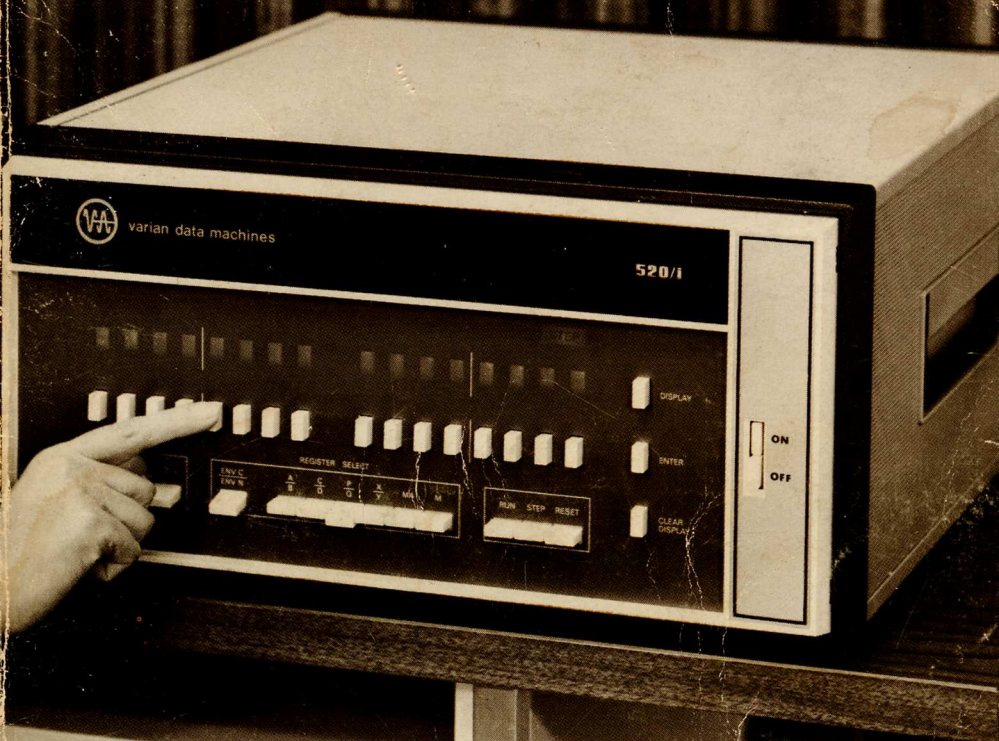




varian
520/i
computer
handbook



varian
data machines

The Big Company in Small Computers

varian
520/i
computer
handbook

TABLE OF CONTENTS

<u>Section</u>	<u>Page</u>
1 INTRODUCTION	1-1
Dual Environment	1-1
Up to 32,768 Memory Bytes	1-2
Variable Word Length	1-2
Multi-Level Addressing	1-2
Over 50 Basic Instructions	1-2
A Choice of 8- or 16-Bit I/O	1-2
Priority Interrupt System	1-4
Operating Controls	1-4
Hardware Options	1-4
Data Communications	1-4
Hexadecimal Arithmetic	1-5
2 VARIAN 520/i SPECIFICATIONS	2-1
3 CENTRAL PROCESSOR UNIT	3-1
Register and Bus Structure	3-1
Dual Environment	3-1
Other Registers	3-5
4 CORE MEMORY	4-1
Control Modes	4-1
Physical Description of Memory	4-1
Interface Lines	4-2
5 WORD FORMATS	5-1
Data Words	5-1
Instructions	5-1
Input/Output Words	5-1

TABLE OF CONTENTS (Cont)

<u>Section</u>	<u>Page</u>
6 ADDRESSING MODES	6-1
Direct Addressing	6-1
Indirect Addressing	6-1
Indexing	6-2
7 INSTRUCTIONS	7-1
8 INPUT/OUTPUT BUS	8-1
I/O Bus	8-1
Line Drivers and Receivers	8-3
I/O Instructions	8-3
Typical Controller Circuits	8-6
9 PRIORITY INTERRUPT SYSTEM	9-1
Interrupt System Instructions	9-4
Interrupt Subroutine Instructions	9-5
Logic Levels on Interrupt Lines	9-5
10 DIRECT MEMORY ACCESS	10-1
Data Rates	10-1
DMA Options	10-2
Internal DMA Logic	10-2
DMA Operating Modes	10-2
DMA Operation With Interrupts	10-4
User-Designed DMA Board	10-5
Power Requirements	10-5
Interface Data, Direct to Internal Bus Structure	10-8
Interface Data, Varian-Supplied Drivers and Receivers	10-14
Timing Data	10-14
11 OPERATOR'S CONSOLE	11-1

TABLE OF CONTENTS (Cont)

<u>Section</u>	<u>Page</u>
12 TELETYPE AND CONTROLLER	12-1
Functional Description	12-1
Teletype Instructions	12-5
Programming Examples	12-7
Physical Description	12-8
13 PAPER-TAPE PUNCH/READER	13-1
Paper Tape Punch	13-1
Punched Paper Tape Reader	13-2
Power Control	13-3
Interface Connections	13-3
Punch/Reader Combinations	13-3
14 OPERATION	14-1
Program Assembly	14-1
Checking the Program	14-1
15 BASIC BOOTSTRAP AND BINARY LOADER, Identification No. 92A0303-001A	15-1
Features	15-1
Loading the Basic Bootstrap	15-1
Loading the Binary Loader	15-2
Operating the Binary Loader	15-2
16 SYMBOLIC ASSEMBLER (Version A, Absolute Object) Identification No. 92A0303-002	16-1
Features	16-1
Preparation of Source Program	16-3
Assembler Pseudo Operations	16-4
BSS Block Storage Specification	16-4
ORG Set Location Counter	16-5
SET Set Symbolic Value	16-5

TABLE OF CONTENTS (Cont)

<u>Section</u>	<u>Page</u>
EQU Equate Symbols	16-5
END End Assembly	16-6
LIT Begin Literal Block	16-6
MOR Halt for More Input	16-6
IOR Begin Indirect Pointer Block	16-7
DA1, DA2, DA3, DA4, Define Constants	16-7
Assembler Output	16-8
Assembler Addressing	16-8
Assembler Operation	16-9
 17 SYMBOLIC ASSEMBLER (Version B, Relocatable Object)	
Identification No. 92A0303-004	17-1
 Features	17-1
Preparation of Source Program	17-2
Assembler Pseudo Instructions	17-2
ENT - Define Entry-Point Symbol	17-2
EXT - Define External Symbol	17-3
Assembler Output	17-3
Assembler Addressing	17-4
Assembler Operation	17-4
 18 RELOCATING AND LINKING LOADER	
Identification No. 92A0203-007	18-1
 Features	18-1
Loader Input	18-1
Object-Type Input	18-3
Sense Switch Settings	18-3
Operation of Loader	18-3
Directive Codes	18-6
Loader As a Callable Subroutine	18-8
 19 MATHEMATICAL SUBROUTINES	19-1
 Features	19-1
General Description	19-3
Label Format	19-4

TABLE OF CONTENTS (Cont)

<u>Section</u>	<u>Page</u>
Number Formats	19-4
Operating Requirements	19-5
Assembly of Math Subroutines with Symbolic Assembler, Version A	19-5
Assembly of Math Subroutines with Symbolic Assembler, Version B	19-8
Subroutine Descriptions	19-8
20 SOURCE TAPE CORRECTION (STC) PROGRAM	
Identification No. 92A0203-005	20-1
Features	20-1
Operating the STC Program	20-1
Input Commands	20-3
Editing Commands	20-4
Output Commands	20-5
Line Number References	20-6
Other Commands	20-6
21 GENERAL DEBUGGING (GAID) Identification No. 92A0203-004	21-1
Features	21-1
Basic GAID Commands	21-2
Construction of Patch Areas	21-6
Special Displays	21-7
Overlaying	21-8
Operation of GAID	21-8
22 BASIC DEBUGGING AID (BAID) Identification No. 92A0203-0036	22-1
Features	22-1
Basic BAID Commands	22-2
Overlaying BAID	22-4
Operation of BAID	22-4
23 BASIC COMPUTER TEST PROGRAM Identification No. 92A0203-005A	23-1
Features	23-1
Executive Routine	23-2
Instruction Tests	23-4

TABLE OF CONTENTS (Cont)

<u>Section</u>	<u>Page</u>
Instruction Test (INST)	23-4
One Byte Non-memory Reference Instruction Test, Part A (INSA)	23-4
One Byte Non-memory Reference Instruction Test, Part B (INSB)	23-8
Two Byte Non-memory Reference Instruction Test (INSC)	23-8
Two Byte Memory Reference Instruction Test (INSD)	23-9
Teletype Tests	23-11
Keyboard Input/Teletypewriter Output Test (TTYA)	23-11
Paper Tape Reader/Punch Test (TTYB)	23-11
Teletypewriter Rotating Pattern Test (TTYC)	23-12
Memory Test	23-13
 24 PERIPHERAL OPERATING SYSTEM (POSY)	 24-1
Features	24-1
Preparation of POSY Test Programs	24-7
Operating the POSY System	24-10
I/O Interface With Peripherals	24-18
 25 INSTRUCTION ADDRESS BOUNDARY TEST	 25-1
Operating the Instruction Address Boundary Test	25-1
Procedure for Determining Type of Error	25-4
 26 INSTALLATION AND INTEGRATION	 26-1
System Layout and Planning	26-1
System Installation	26-3
Teletype	26-3
Mainframe	26-6
Expansion Chassis	26-9
Memory Expansion and Cabling	26-12
8-Bit I/O Cables and Controllers	26-12
16-Bit I/O Cables and Controllers	26-18
Interrupt Connections	26-19
Power Supplies	26-20

TABLE OF CONTENTS (Cont)

<u>Section</u>	<u>Page</u>
27 16-BIT PERIPHERAL ADAPTER	27-1
Varian 620/i Controllers	27-1
Functional Description	27-3
F Register	27-6
Timing	27-7
Physical Construction	27-9
28 MEMORY PARITY OPTION	28-1
Parity Error Subroutine	28-1
Parity Diagnostic Routine	28-3
Operating the Parity Diagnostic Routine	28-5
Error Halts	28-5
29 POWER FAIL/RESTART ROUTINE	29-1
Functional Description	29-1
30 REAL-TIME CLOCK OPTION	30-1
Functional Description	30-1
Programming the Real-Time Clock	30-3
Physical Description	30-4
31 VARIAN 520/DC DATA COMMUNICATION SYSTEM	31-1
System Description	31-1
Varian 520/i-60 Controller	31-3
Software	31-5
Control Stacks	31-6
System Specifications	31-6
APPENDIX	A-1

LIST OF ILLUSTRATIONS

<u>Figure</u>		<u>Page</u>
1-1	Varian 520/i Computer	1-3
3-1	Block Diagram, Varian 520/i Computer System	3-3
3-2	Block Diagram, Varian 520/i Central Processor Unit (CPU)	3-4
4-1	Varian 520/i Full-Cycle Memory Timing	4-3
5-1	Varian 520/i Data Word Formats	5-2
5-2	Varian 520/i Instruction Formats	5-3
5-3	520/i Input/Output Word Formats	5-4
6-1	Address Bit Decoding	6-2
6-2	Memory Sector Allocation	6-3
7-1	Two-Byte Memory-Reference Instructions	7-2
7-2	One-Byte Memory-Reference Instructions	7-4
7-3	Two-Byte Non-Memory-Reference Instructions	7-7
7-4	Three-Byte Non-Memory-Reference Instructions	7-10
7-5	Two-Byte Input/Output Instructions	7-11
7-6	Teletype Controller Reserved Instructions	7-12
7-7	Punched-Tape System Reserved Instructions	7-13
8-1	Typical Varian 520/i Data Bus Line	8-2
8-2	Typical Control or Address Line from the Varian 520/i	8-4
8-3	Typical Control Line to the Varian 620/i	8-5
8-4	Typical Varian 520/i Sense Logic	8-8
8-5	Typical Varian 520/i Function Control Logic	8-9
8-6	External Sense Timing	8-10
8-7	External Function Control Timing	8-10
8-8	Typical Varian 520/i Data Transfer Output Logic	8-11
8-9	Typical Data Transfer Input Logic	8-13
8-10	Data Output Timing Requirements	8-14
8-11	Data Input Timing Requirements	8-14
9-1	Interrupt Configuration	9-2
9-2	Varian 520/i Priority Interrupt System	9-3
9-3	Typical Interrupt Circuits	9-6
10-1	DMA Internal Logic	10-3
10-2		10-6
10-3		10-7
10-4	Pin Assignments, Connector J5	10-9

LIST OF ILLUSTRATIONS (Cont)

<u>Figure</u>		<u>Page</u>
10-5	DMA Signals To and From the Main Frame	10-10
10-6	DMA Address Connections to the Varian 520/i Mainframe	10-12
10-7	DMA Data To and From the Varian 520/i Mainframe	10-13
10-8	Plug-Pin Assignments, Varian 520/i-12 DMA Option	10-15
10-9	Signal Functioning, Varian 520/i DMA Option	10-16
10-10	DMA Port Line Driver/Receiver Block Diagram	10-17
10-11	DMA Port Timing	10-18
11-1	Varian 520/i Front Panel	11-2
12-1	Hexadecimal Representation of all Acceptable ASCII Characters	12-2
12-2	Block Diagram of the Computer Interface to the Teletype Controller	12-3
12-3	Block Diagram of the Teletype Controller and Teletype	12-4
12-4	Input/Output Instructions	12-6
13-1		13-2
13-2		13-4
13-3		13-5
14-1	Basic Bootstrap Program Loading Steps	14-2
14-2	Binary Loader Program Loading Steps	14-3
14-3	Binary Object Program Operating Steps	14-3
14-4	Source Statement Coding Rules	14-4
14-5	Preparation for Assembly	14-5
14-6	Pass 1 Operation	14-6
14-7	Pass 2 and 3 Operation	14-9
14-8	Sequential Memory Access Steps	14-10
15-1	Basic Bootstrap Program	15-2
15-2	Basic Bootstrap and Binary Loader Memory Map	15-3
18-1	520/i Relocating and Linking Loader Memory Map	18-2
19-1	Index of Mathematical Subroutine Descriptions	19-2
19-2	Matrix of 520/i Floating Point Math Subroutine Size and Interaction	19-6
19-3	Summary of 520/i Math Subroutine Size and Timing	19-7
19-4	Flowchart of EX	19-28
19-5	Flowchart of LN	19-32
19-6	Flowchart of QF	19-35
19-7	Flowchart of FQ	19-38
19-8	Flowchart of SI	19-40
19-9	Flowchart of CO	19-43
19-10	Flowchart of AT	19-44
20-1	Summary of STC Commands	20-2

LIST OF ILLUSTRATIONS (Cont)

Figure		Page
21-1	Table of Commands for GAID	21-3
21-2	Varian 520/i General Debugging Aid (GAID) Memory Map	21-9
22-1	Table of Commands for GAID	22-3
22-2	Basic Debugging Aid (BAID) Memory Map	22-5
23-1	Mnemonics for Individual Tests	23-3
23-2	Error Messages	23-5
23-3	Hexadecimal Representation of all Acceptable ASCII Characters	23-10
24-1	Summary of POSY Operation Codes	24-2
24-2	POSY Symbolic Names for Available Peripherals	24-3
24-3	POSY Symbolic Characters for I/O Modes	24-3
24-4	Symbolic Character Designations for Data Generators	24-4
24-5	POSY CONTROL Statement Command Fields	24-8
24-6	Identification Numbers of POSY Object Tapes	24-11
24-7	POSY Halt Codes and Their Descriptions	24-12
24-8	Summary of POSY Error Messages Occurring During Compilation	24-17
25-1	Memory Map, Instruction Address Boundry Test	25-2
25-2	Contents of Typical Test Block	25-3
26-1	Side View of Typical System Installation	26-2
26-2	Cable Connections, Front of System (Front Panels Open)	26-4
26-3	Cable Connections, Rear of System	26-5
26-4	Outline Drawings of Mainframe and Expansion Chassis	26-7
26-5	Tray Locations, Mainframe	26-8
26-6	Tray Locations, Expansion Chassis with 16K Memory	26-10
26-7	Tray Locations, Expansion Chassis with Up to 16 I/O Controllers	26-10
26-8	Typical Tray Locations, Expansion Chassis with Both Memory and I/O Controllers	26-11
26-9	Signal/Power Distribution Boards on Expansion Chassis	26-11
26-10	Block Diagram of Expanded Memory	26-13
26-11	Expanded Memory Connector Board	26-14
26-12	Pin Assignments, Memory Connectors	26-15
26-13	Terminator Board for Memory Bus	26-16
26-14	Block Diagram, 8-Bit I/O Expansion Chassis	26-16
26-15	I/O Connector Board	26-17
26-16	Card Connector Locations on I/O Interface Board	26-17
26-17	Pin Assignment, Connector J2, I/O Interface Board	26-18
26-18	Pin Assignment, Connector J3, I/O Interface Board	26-19
26-19	8-Bit I/O Bus	26-20
26-20	4-Section I/O Controller Tray	26-21

LIST OF ILLUSTRATIONS (Cont)

<u>Figure</u>		<u>Page</u>
26-21	Drivers and Receivers, I/O Controller Tray	26-22
26-22	Block Diagram, 16-Bit I/O Expansion Chassis	26-23
26-23	16-Bit Peripheral Chassis Adapter	26-24
26-24	16-Bit I/O Bus Cabling	26-24
26-25	Block Diagram, Interrupt Connection	26-25
26-26		26-26
26-27		26-26
26-28	Power Requirements for Varian 520/i System Components	26-27
26-29	On-Off Control and Expansion — Chassis Power Supplies	26-28
27-1	Pin Assignments, I/O (E-Bus) Cable Connector, 16-Bit Peripheral Adapter	27-2
27-2	Device Addresses for Varian 620/i Controllers	27-4
27-3	Relationship Between 8-Bit and 16-Bit Channels	27-4
27-4	16-Bit Peripheral Adapter Block Diagram	27-5
27-5		27-6
27-6	Comparison of Peripheral Adapter and Varian 620/i Timing	27-8
27-7	16-Bit Peripheral Adapter Waveforms (Referenced to Varian 520/i Clock)	27-9
28-1	Memory Parity Option Block Diagram	28-2
28-2	Memory Map, Parity Diagnostic Routine	28-4
29-1	Timing Diagrams, Power Fail/Restart Routine	29-2
29-2	Memory Map, Power Fail/Restart Routine	29-2
30-1	Real-Time Clock Block Diagram	30-2
31-1	Varian 520/DC Data Communication System	31-2
31-2	Varian 520/i-60 Data Communication Controller	31-4

SECTION 1 – INTRODUCTION

The Varian 520/i is a general-purpose digital computer with a number of design features that make it unique in its class.

Included are such special capabilities as dual processing environments, variable word length, direct addressing to 4,096 bytes, a choice of 8- and 16-bit input/output transfers, and a versatile, four-level priority interrupt system. These combine to make the Varian 520/i an efficient and powerful machine for a wide range of communication, process-control, and data-acquisition applications.

The purpose of this Handbook is to introduce the reader to the Varian 520/i and, at the same time, serve as a reference document for individuals who are using the computer or incorporating it into a larger data-processing system.

To facilitate these dual objectives, the discussion has been divided into sections, each of which is a self-contained description of a particular hardware or software element. Items that are common to a number of subjects, such as hexadecimal conversion tables and detailed instruction descriptions, are grouped together in an Appendix where they can be readily found, both on first reading and for later reference.

This introductory section is designed as an overview of both the computer and the

material contained in the sections that follow.

Dual Environment

One of the most striking features of the Varian 520/i is the presence of two processing environments within the central processor unit (CPU).

Section 3 describes how the CPU is provided with two sets of accumulators, program counters, and index registers. Separate programs can be written for operation in each of the two environments; the computer can switch from one program to the other (e.g., from a main program to an I/O subroutine) without disturbing the contents of the registers in either environment. There is no time lost by time-consuming save-and-restore routines; the computer can return to the first environment and proceed as if no interruption had taken place.

The dual-environment arrangement is of special value in applications requiring extensive I/O activity, such as in data-communication systems. A background program is written to handle the routine data processing, while a foreground program, using a separate environment, performs the input/output operations quickly and efficiently.

Up to 32,768 Memory Bytes

Varian Data Machines has specialized, through the years, in the design of high-speed, low-error-rate core memories. This fact is reflected in the description of the Varian 520/i memory given in Section 4.

A complete 4,096-byte memory bank is contained within a single circuit-board tray. Two such trays fit within the basic main-frame chassis. Up to eight can be included in a system through the addition of expansion chassis. All 32,768 bytes in a fully expanded memory are addressable through instructions provided with the basic computer.

Variable Word Length

The Varian 520/i is designed essentially as an 8-bit machine, again making it appropriate to a range of data-communication applications. Internally, however, the Varian 520/i can process data as if the word length were 8, 16, 24, or 32 bits. The word length is set by software at 1, 2, 3, or 4 bytes, 8 bits to a byte. The accumulator in each of the two environments is capable of holding a full 32-bit word.

The variable word length is of particular value in instrumentation and data acquisition systems where varying precision requirements must be met. The precision can, in effect, be set to the requirements of the data, not the data to the requirements of the machine. The variable data-word formats are defined in Section 5.

Multi-Level Addressing

Few computers in the Varian 520/i class offer such a range of memory-addressing

techniques. Section 6 outlines the modes that are available.

Direct addressing is possible to 4,096 bytes, representing the total memory in a computer of minimum configuration. Several types of indexed and indirect addressing are also available to the programmer, giving him access to the 32K-bytes of a full-configuration computer in only three machine cycles.

Over 50 Basic Instructions

A major strength of the Varian 520/i is the repertoire of instructions presented in Section 7 and detailed in the Appendix.

The instructions vary in length from one to three bytes and include such special commands as a single instruction to change from one environment to another.

All instructions are provided with mnemonic codes that can be used by the programmer to prepare a symbolic program with easily understood notations. The Varian 520/i Assembler converts these symbols into an object program expressed in machine language.

A Choice of 8- or 16-Bit I/O

The basic I/O link for the Varian 520/i is an 8-bit bus that can address up to 32 peripheral controllers. Details on this bus are given in Sections 8 and 26.

In addition, through the addition of the 16-bit Peripheral Adapter described in Section 27, the systems designer can add

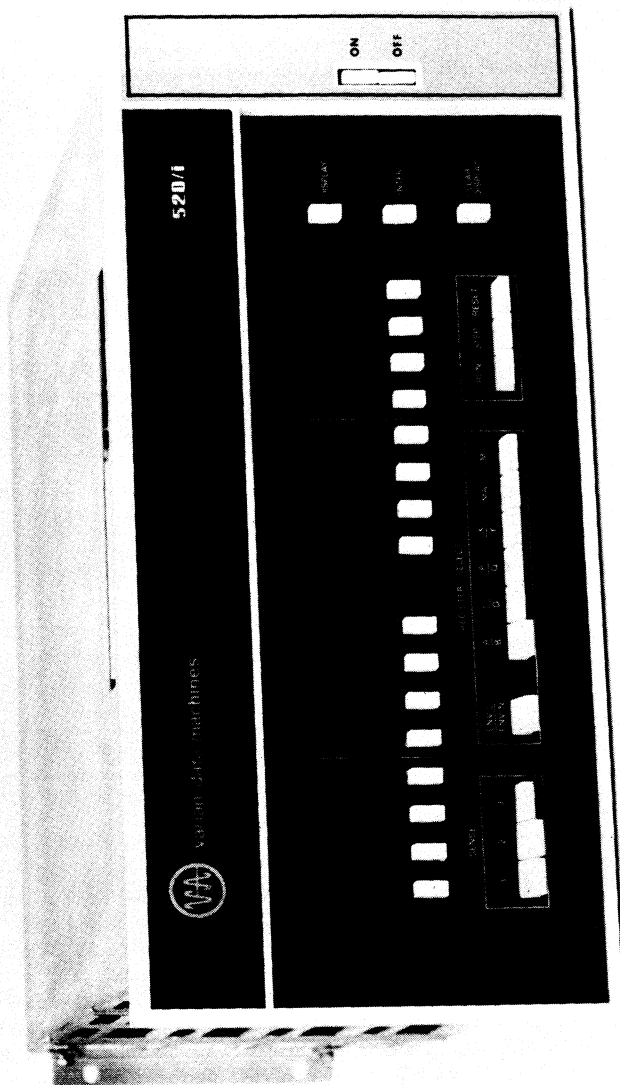


Figure 1-1. Varian 520/i Computer

introduction

16-bit controllers and devices, such as analog-to-digital converters and CRT displays. Controllers for the 16-bit bus are the field-tested units designed originally for Varian 620/i computer systems.

Still a third I/O technique is available through the Direct Memory Access (DMA) facility defined in Section 10. The DMA memory port allows data transfers to occur at rates up to 666,000 bytes per second.

Priority Interrupt System

Eleven external interrupt lines, arranged in four levels of priority, are a standard feature of the Varian 520/i. Sections 9 and 26 discuss the use of these interrupts in systems applications. In a typical instance, interrupts from a peripheral device are used to switch the program from one environment to another. An I/O subroutine transfers a byte of data between the computer and the peripheral device and then returns the program to the first environment.

The interrupt system includes provisions for enabling and disabling the interrupts, either individually or as a block.

Operating Controls

The Varian 520/i is provided with simple, human-engineered controls on the front panel for program entry and status checking. In addition, an ASR Teletype is included in most Varian 520/i systems to serve as the principal communication device between operator and computer. Details on both of these elements are presented in Sections 11 and 12.

Software

A full complement of programming and diagnostic aids are provided with the Varian 520/i. Included are bootstrap and binary-object-tape loading routines, two types of symbolic assemblers, an extensive set of mathematical subroutines, program-debugging aids, and hardware test programs.

Detailed descriptions on all of these software packages are given in Sections 15 through 25.

Hardware Options

The value of a computer must be measured in part by the "extras" that can be added to meet special systems requirements. A number of such hardware options are available for use with the Varian 520/i. Typical are the memory-parity, power failure/restart, and real-time clock options defined in Sections 28, 29, and 30.

Data Communications

The special assets of the Varian 520/i in data-communications applications are reflected in the design of the Varian 520/DC system described in Section 31.

The Varian 520/DC is designed specifically for interactive time sharing systems involving a frequent interchange of relatively short messages. The system can serve either as a data concentrator or as a computer pre-processor.

Hexadecimal Arithmetic

All four Varian 520/i word lengths can be conveniently expressed in hexadecimal (base 16) form. To convert to this form, the binary word, 8, 16, 24, or 32 bits in length, is divided into 4-bit segments (0000 to 1111 binary, or 0 to 16 decimal). A single hexadecimal numeral (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, or F) is then used to

represent each 4-bit group. Only two hexadecimal numerals are required, therefore, to represent an 8-bit word; only 8 numerals to represent a 32-bit word.

Hexadecimal notation is signified throughout the Varian 520/i Handbook by a dollar-sign (\$) prefix. To assist the user, an extensive set of hexadecimal-to-decimal conversion tables are included in the Appendix.

SECTION 2 – VARIAN 520/i SPECIFICATIONS

Specifications	Characteristics																
Type	General-purpose digital, designed for on-line data systems.																
Memory	Magnetic core, 8 bits, 1.5 microseconds full cycle, 500 nanoseconds access time, 4096 bytes minimum, expandable to 32,768 bytes.																
Arithmetic	Parallel, binary, fixed point, 2's complement.																
Operand Word Length	8, 16, 24 or 32 bits.																
Addressing	Direct addressing to 4096 bytes. Index with X register in hardware, does not add to execution time. Multi-level indirect addressing.																
Instructions	Over 50 basic instructions with over 500 useful register to register operations. <table> <tr> <th>Number</th><th>Type</th></tr> <tr> <td>1</td><td>Load</td></tr> <tr> <td>1</td><td>Store</td></tr> <tr> <td>2</td><td>Arithmetic</td></tr> <tr> <td>3</td><td>Logical</td></tr> <tr> <td>14</td><td>Skip</td></tr> <tr> <td>3</td><td>Branch</td></tr> <tr> <td>4</td><td>Input/Output</td></tr> </table>	Number	Type	1	Load	1	Store	2	Arithmetic	3	Logical	14	Skip	3	Branch	4	Input/Output
Number	Type																
1	Load																
1	Store																
2	Arithmetic																
3	Logical																
14	Skip																
3	Branch																
4	Input/Output																

Specifications	Characteristics									
Registers available to the programmer		<table><tr><th>Number</th><th>Type</th></tr><tr><td>8</td><td>Register Change</td></tr><tr><td>8</td><td>Logical Shift</td></tr><tr><td>16</td><td>Control</td></tr></table>	Number	Type	8	Register Change	8	Logical Shift	16	Control
	Number	Type								
	8	Register Change								
	8	Logical Shift								
	16	Control								
	P Register:	Program counter in current environment, 16 bits								
	X Register:	Index register in current environment, 16 bits								
	A Register:	Most-significant accumulator in current environment, 16 bits								
	C Register:	Least-significant accumulator in current environment, 16 bits								
	OPC Register:	Operand precision (word length) register in current environment, 2 bits								
	OVC Register:	Overflow register in current environment, 1 bit								
	Q Register:	Program counter in noncurrent environment, 16 bits								
	Y Register:	Index register in noncurrent environment, 16 bits								
B Register:	Most-significant accumulator in non-current environment, 16 bits									
D Register:	Least-significant accumulator in noncurrent environment, 16 bits									
OPN Register:	Operand precision register in non-current environment, 2 bits									

Specifications	Characteristics
Registers not available to the programmer	OVN Register: Overflow register in noncurrent environment, 1 bit Z Register: Pseudo-register containing only ZEROs F Register: I/O channel register in optional peripheral controller, 16 bits I Register: Instruction register, 8 bits S Register: Temporary operand storage register, 8 bits MB Register: Memory buffer, temporary operand storage register, 8 bits MA Register: Memory address register, 15 bits
Features	Dual environment: two sets of hardware registers (program counter, index register, accumulator, control registers) eliminate the necessity to save and restore register contents when transferring between program routines in different environments.
INPUT/OUTPUT	
Processor	Programmed data transfer: single word to/from accumulator External control lines External sense lines
Automatic Data Transfer	Direct-memory-access facility with data rates up to 666,000 bytes per second.
Priority Interrupts	Eleven interrupts are standard with group enable/disable and individual arm/disarm.

Varian 520/i specifications

Specifications	Characteristics
Console	Display and data entry switches for all operational registers and memory, single step control.
PHYSICAL	
Dimensions	Main frame: 8-3/4 inches high, 19 inches wide, 20-3/4 inches deep.
Weight	Main frame with internal power supply: 48 pounds
Power	115 vac $\pm$ 10 vac, 60 Hz, 2-6 amps. Power supplies are regulated. Additional regulation is not required with normal commercial power sources. Conversion for European power available at added cost.
Expansion	Main frame contains provisions and space for 8,192 bytes of memory, teletype controller, paper tape system controller and 16-bit I/O channel.
Installation	Mounts in standard 19-inch cabinet. No air conditioning, sub-flooring, special wiring or site preparation are required.
Environment	0°C to 50°C, 0 to 90% relative humidity.
Mainframe	Integrated circuit, 8-MHz clock, logic levels are 0 vdc false (ZERO) and +5 vdc true (ONE).

SECTION 3 – CENTRAL PROCESSOR UNIT

The central processor unit (CPU) contains all the Varian 520/i computer circuit elements except the memory, the control panel, and the controller circuits for peripheral devices.

This relationship is illustrated in Figure 3-1, a simplified block diagram of a Varian 520/i system. A minimum computer configuration would consist of the central processor with control panel, 4096-bytes of memory, 8-bit I/O bus, and a teletype with its associated controller.

Optional additions would include an expanded memory (up to 32K bytes in 4K-byte increments), additional 8-bit peripheral devices, a 16-bit I/O bus with a choice of 16-bit controllers and devices, and a high-speed direct memory access (DMA) channel.

Each of these other elements, other than the CPU, are described in detail elsewhere in this handbook.

Register and Bus Structure

The internal structure of the central processor or CPU is shown in Figure 3-2. The basic elements are 16 integrated-circuit registers, each capable of storing 1, 2, 8, or 16 bits, and 7 internal busses that distribute data in 8-bit bytes. Not shown is the basic computer clock, operating at a 4 MHz rate

(250 nanosecond intervals), and the associated timing and control circuits.

All data transfers and arithmetic operations within the CPU are 8-bit parallel with a machine cycle time of 1.5 microseconds. The precision of these operations can, however, be set, under software control, at 8, 16, 24, or 32 bits. The CPU responds by processing 1, 2, 3 or 4 bytes in sequence while executing a single instruction. When the operation is a data transfer to and from memory the memory address of the first byte serves as the address for the complete multiple-byte word.

Dual Environment

Twelve of the sixteen IC registers are accessible to the programmer, either through software or directly from the control panel.

The twelve registers are divided into two parallel groups, each consisting of the following elements:

Program Counter	16 bits
Index Register	16 bits
Accumulator:	
Most Significant Half	16 bits
Least Significant Half	16 bits

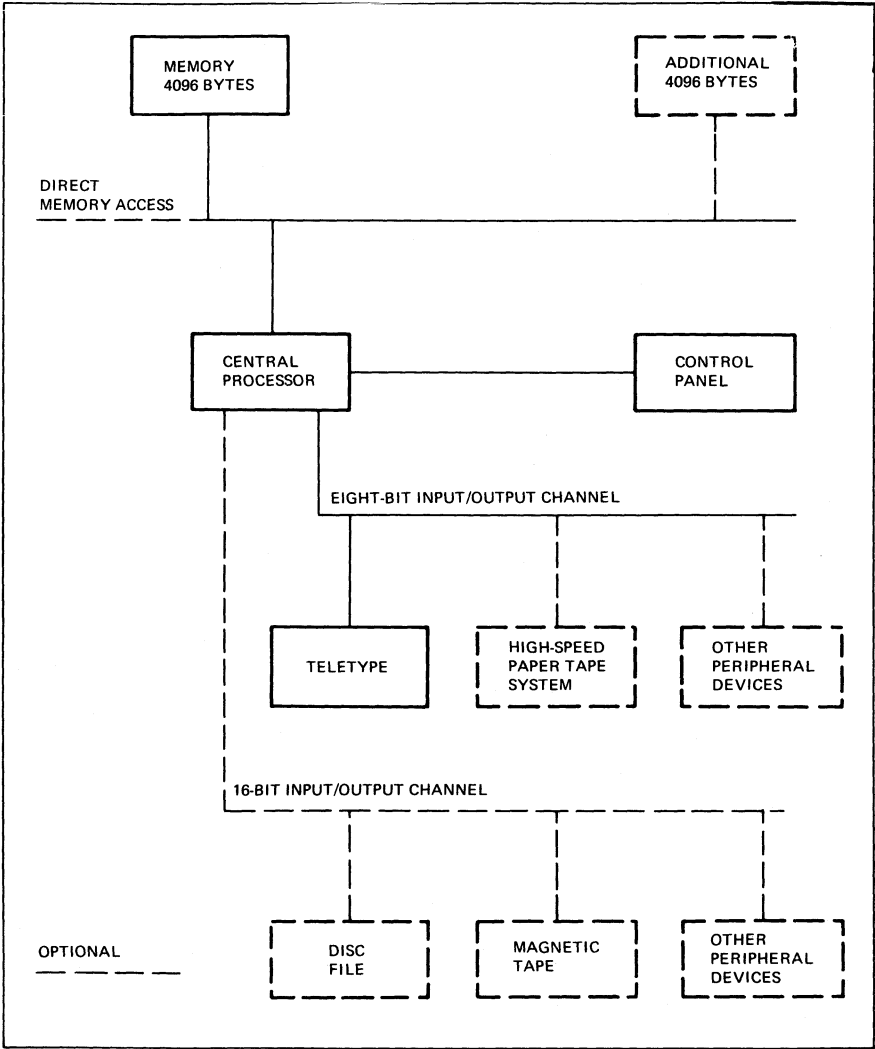


Figure 3-1. Block Diagram, Varian 520/i Computer System

Overflow Indicator (not shown) 1 bit

Operand Precision (not shown) 2 bits

The two accumulator registers operate together as a single 24-bit or 32-bit accumulator when the precision is set at 3 or 4 bytes.

Each of these sets of registers forms a processing "environment." Transfer between one environment and the other can be accomplished by a single program instruction. The contents of the registers of the first environment remains unchanged; consequently there is no time wasted for saving or re-storing of data when the program is interrupted, for example, to process an I/O request from a peripheral device.

One environment is interruptable; the other is non-interruptable. To process an I/O interrupt, for example, the main program would be written for the interruptable environment and the I/O subroutine written for the non-interruptable environment. The occurrence of an interrupt would initiate a program switch to the non-interruptable environment for execution of the I/O subroutine. The last step in the subroutine would be a programmed return to the interruptable environment and a resumption of the main program.

The transfers between environments are accomplished in a single machine cycle, or 1.5 microseconds.

The environment in which the computer is operating is designated as the "current" environment; the other is designated the "non-current" environment. Either the interruptable or the non-interruptable environment can be current or non-current, depending on the state of the program. Programs normally start in the interruptable environment, branching to the non-interruptable environment only for subroutines that return the program to the interruptable. A system reset returns the system to the interruptable environment.

All front-panel indicators and instruction symbols relating to the two environments are based on their current/non-current status, rather than their interruptable/non-interruptable differentiation, with the single exception of the two overflow indicators for the interruptable environment (O₁) and the non-interruptable environment (O₂).

Thus the A Register, as symbolized on the front panel and in register-transfer instructions, is the 16-most significant bits in the accumulator in the current environment. It could be in either the interruptable environment or the non-interruptable environment at any given moment, depending on the status of the program.

Stated another way, the most-significant accumulator in the interruptable environment, for example, could be either an A Register or a B Register, at any given moment, depending on whether the interruptable environment were current or non-current.

This is the reason why the environmental registers shown in Figure 3-2 do not have

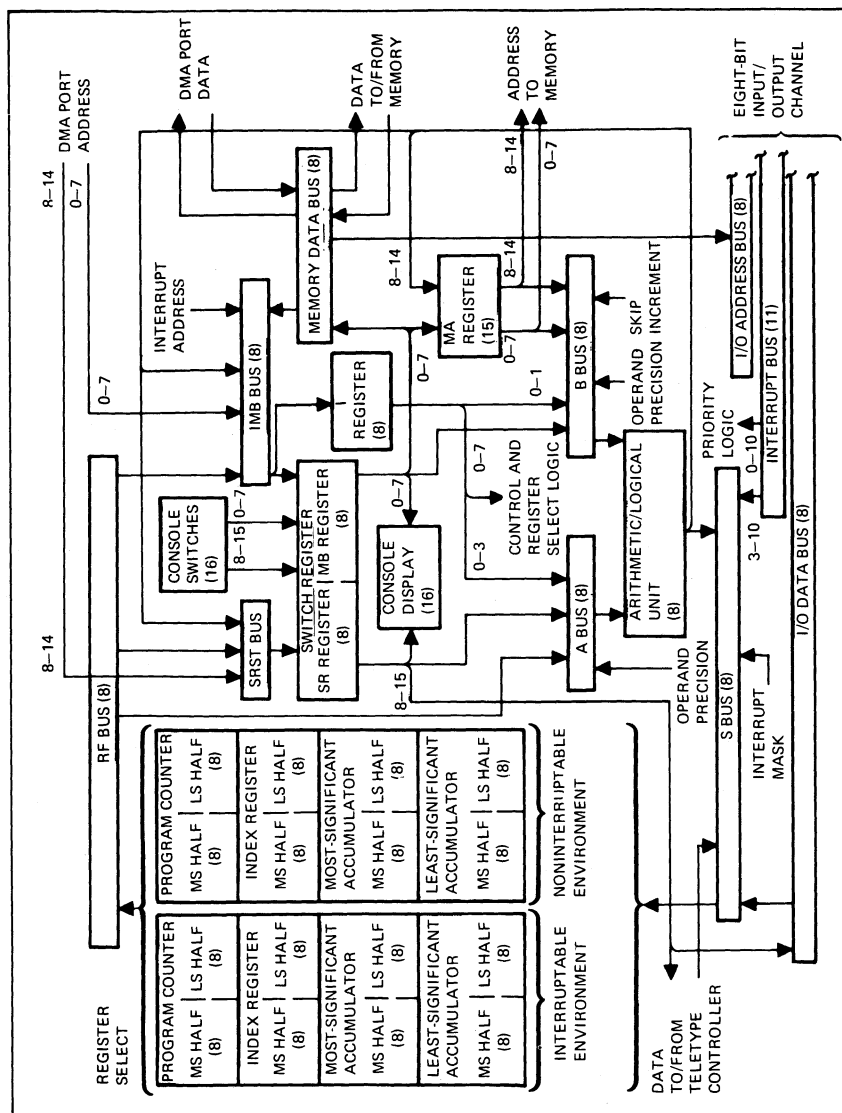


Figure 3-2. Block Diagram, Varian 520/i Central Processor Unit (CPU)

identifying letters. Their instruction symbology changes with their environmental status.

Other Registers

There are four registers that are not part of the two environments and are not accessible to the programmer:

I Register is an 8-bit instruction register that holds the current instruction and address modification.

SR Register is an 8-bit temporary storage for presenting data to the arithmetic unit, for control-panel entry and display, and for data output on the I/O bus.

MB Register is an 8-bit temporary storage for presenting data to the arithmetic unit, for control-panel entry and display, and for accessing memory.

MA Register is a 16-bit register that holds the current operand address.

SECTION 4 – CORE MEMORY

The Varian 520/i computer has a random-access core memory that is configured in memory stacks each capable of storing 4,096 bytes, with 8 or 9 bits per byte. The memory is expandable within the main-frame chassis bytes to 8,192 bytes. Additional 4,096-byte increments allow expansion to a maximum of 32,768 bytes.

The memory electronics consist of nine data loops and sufficient address circuitry to interrogate 4,096 byte locations. The storage element consists of a magnetic core plane consisting of 8 or 9 mats of 4,096 cores each.

The digit and address drive sources for the memory are an integral part of the computer power supply. The drive sources track a thermistor network that senses the temperature of the core plane and adjusts accordingly.

The cycle time is 1.5 microseconds with an access time of 500 nanoseconds.

Control Modes

All control modes are random access and can occur in any sequence. The modes are determined by two control levels: $\overline{CW}$ and CW.

Read/Restore ($\overline{CW}$). The data in the address location specified by the address inputs are

read out and placed in the memory register. The data that was entered into the data register are then written into the address register originally specified by the address inputs.

Clear/Write (CW). The data in the address location, specified by the address inputs, are cleared to all zeros and the data to be written are strobed into the memory register. The data that was strobed into the data registers are then written into the address registers originally specified by the address inputs.

Physical Description of Memory

Each 4,096-byte memory stack consists of four circuit-board elements.

The Data and T/C board consists of nine sense amplifiers, nine data registers, nine output data drivers, a threshold regulator, and the required timing and control logics.

The Address board consists of address buffers, first order address decoding, and 32 driver/switch pairs.

The Inhibit board consists of nine digit driver circuits.

The Core Plane board consists of the second-order diode decode matrices; the 8 or 9 mats of 4,096 30-mil cores strung in a 4-wire, 3-D configuration; and the routing of

sense, inhibit, and address to the three previously described electronics boards.

Interface Lines

Interface lines between the memory and the balance of the computer include address, data out, data in, read and write clocks, clear write/read restore, memory enable, drive source control, and drive source lines.

Address Lines (MA) consist of 15 lines. The 12 least significant lines give the binary number of the memory location to be interrogated within a 4,096-byte increment. The three most significant lines select the 4,096-byte increment. The three most significant lines select the 4,096-byte increment that is to be interrogated. These lines are stable prior to, or equal to, 40 nanoseconds after the leading edge of the read clock and remain stable for at least 710 nanoseconds after the write clock.

Memory Data Output (MDAT) consists of nine lines which describe the information stored in the data registers. These lines are stable 500 nanoseconds after the leading edge of the read clock while in Read/Restore mode. These lines are stable 50 nanoseconds after the leading edge of the write clock while in the Clear/Write mode.

Memory Data Input (MB) consists of nine lines which describe the information to be stored in the data registers. These lines are stable prior to, or equal to, 40 nanoseconds after the leading edge of the write clock while in the Clear/Write mode.

Memory Read Clock (MREAD) initiates a read cycle. The read clock pulse width must be greater than 335 nanoseconds and less

than 415 nanoseconds. During Read/Restore mode this clock reads the contents of the addressed word and stores it in the data register. During the Clear/Write mode this clock clears the contents of the data input lines. The read clock's leading edge cannot follow the write clock's leading edge by less than 710 nanoseconds.

Memory Write Clock (MWRT) initiates a write cycle. The write clock pulse width must be greater than 335 nanoseconds and less than 415 nanoseconds. The write clock writes the contents of the data register into the location specified by the address lines. The write clock's leading edge cannot follow the read clock's leading edge by less than 585 nanoseconds.

Clear Write/Read Restore Modes is a single line that controls whether the operation is either read/restore or clear/write. To perform a read/restore operation, this line must be false (logic "0") and to perform a clear/write operation this line must be true (logic "1"). To perform either operation requires that the memory receive a read and a write clock as previously specified.

Memory Enable Line (DAG) is a single line that controls reception of the read and write clocks. When this line is false (logic "0"), reception of the read or write clocks will be inhibited and vice versa when this line is true (logic "1").

Drive Source Control is a line that connects a thermistor network to the computer power supply for the purpose of varying the drive source level in accordance with the core plane temperature. The drive source tracks the control at a 500 ohm per volt rate.

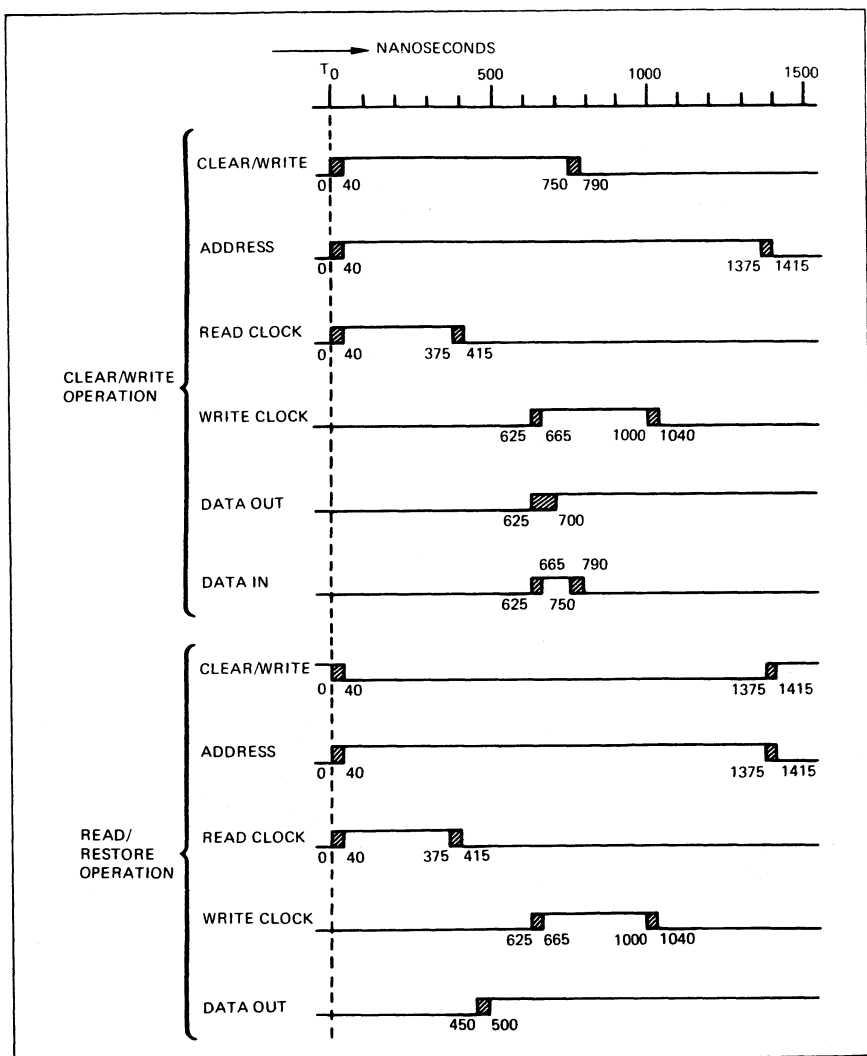


Figure 4-1. Varian 520/i Full-Cycle Memory Timing

Drive Source consists of the digit drive and two address half select drive lines. The two address half select drive sources consist of resistors connected between the address half select lines and the digit drive source. Communication between the drive sources and the memory is via twisted pair.

All signal lines between the memory and the computer are via printed circuitry. Expan-

sion beyond 8,192 words requires an interconnect cable, not exceeding 8 inches in length (see Section 26).

Internal signal line routing is via printed circuitry or single wire jumper with the exception of inhibit and sense signal routing which is via twisted pair.

The timing diagrams in Figure 4-1 describe the memory operations.

SECTION 5 - WORD FORMATS

There are three types of words used in the Varian 520/i. These are:

Data Words stored in memory or in an accumulator

Instructions stored in memory

Input/Output Words transferring data and instructions between the computer and peripheral device.

Each type of word is distinguished by its own distinctive format or formats. These are described in the following paragraphs. Bit positions are numbered, in all cases from right to left, with the highest-numbered bit as the most significant and bit 0 as the least significant.

Data Words

Data may occupy as few as eight bits or as many as 32 bits in memory and in the accumulator. (See Figure 5-1.)

The precision of the data may be set by the program, on a word-by-word basis, as 1, 2, 3, or 4 bytes, representing 8, 16, 24, or 32 bits.

All negative quantities are represented in two's complement form (see page A-2). The most significant bit is zero for positive numbers and one for negative numbers.

Instructions

Instructions occupy as few as eight or as many as 24 bits in memory. Instructions are either memory-reference or nonmemory-reference types as shown in Figure 5-2. In the figure,

- | | |
|-------|--|
| i | is a binary code that indicates the instruction type |
| m | is a binary code that indicates the addressing mode |
| y | is a binary operand address that may be modified to obtain the effective address |
| f | is a binary code that indicates a particular function to be performed |
| f1 | is a binary code that indicates a function class modified by f2 and f3 |
| f2,f3 | are binary codes that specify a particular function within class f1. |

Input/Output Words

Basic communication with the Varian 520/i utilizes an eight-bit data word, an eight-bit address word, and individual control and interrupt signals. Optional 16-bit communication with the Varian 520/i is through a

peripheral adapter that time-shares 16 lines for output functions and bidirectional data.

The Input/Output word formats for the two I/O channels are shown in Figure 5-3.

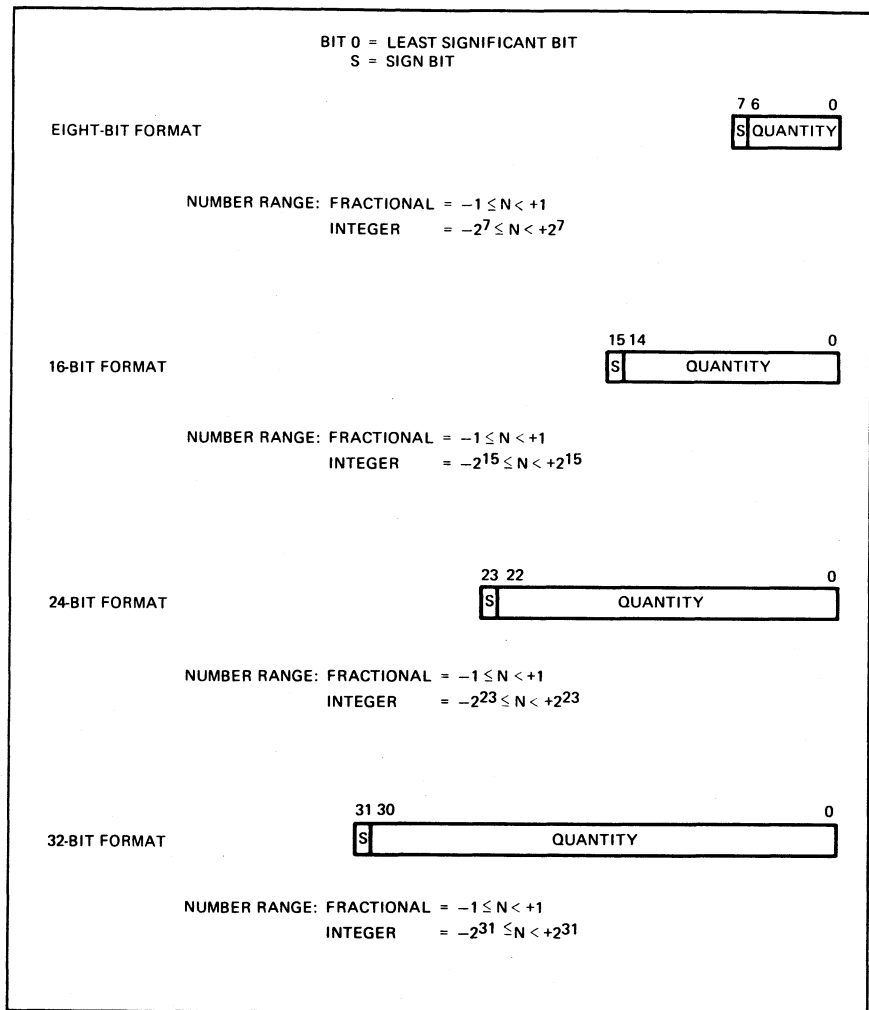


Figure 5-1. Varian 520/i Data Word Formats

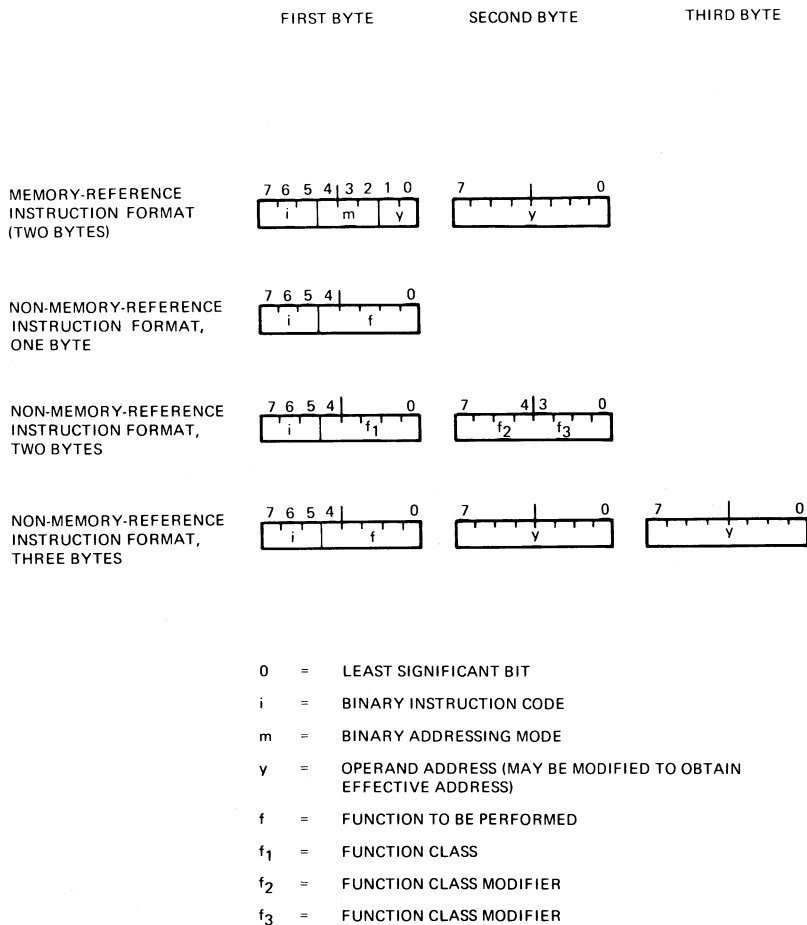


Figure 5-2. Varian 520/i Instruction Formats

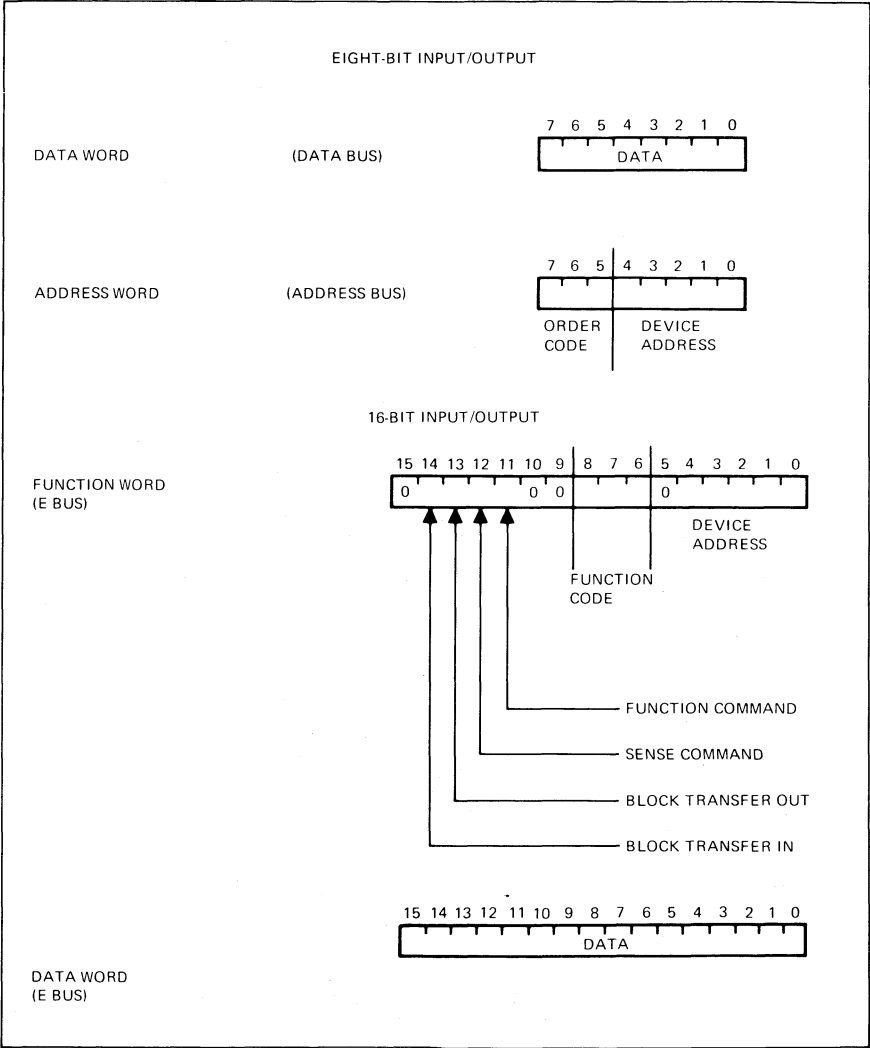


Figure 5-3. 520/i Input/Output Word Formats

SECTION 6 — ADDRESSING MODES

Addressing the contents of memory is a function of the current operand and the accumulator word length. The word length, or precision, is program selectable, and can be set at one, two, three, or four bytes, representing 8, 16, 24, or 32 bits.

The memory is addressed at the byte level. If the current precision is set at two, three or four bytes, the operand address refers to the memory location of the most-significant byte. The accessing of multiple bytes of memory occurs automatically in the correct order for execution.

Address modification is defined by the m field of the instruction word (see Word Format Section). The three bits in this field specify direct, indirect or indexed addressing. The bits are decoded by the central processor into the eight address conditions defined in Figure 6-1.

The sectors referred to in the address conditions are sections of memory, each with a capacity of 1,024 bytes. The basic Varian 520/i has a memory consisting of four sectors. This can be expanded to a maximum of 32 sectors through additional memory modules.

The numbering of the sectors is summarized in Figure 6-2.

Direct Addressing

There are two forms of direct addressing used in the Varian 520/i:

Direct to lower four sectors enables direct addressing to 4096 bytes of memory. For the standard computer this is total direct addressing.

Direct to current sector gives access to 1024 bytes in the sector in which the current instruction is being executed (determined by the most-significant bits of the program counter).

Indirect Addressing

Indirect addressing uses the address y to obtain another operand address contained as a 16-bit word in memory locations y and y + 1. A 16-bit address is always assumed when indirect addressing is specified. The most-significant bit specifies further indirect addressing when it is set. Each level of indirect addressing requires an additional two machine cycles of execution time.

The Varian 520/i uses two modes of indirect addressing:

Indirect addressing to zero sector. Regardless of the current sector, the addressing mechanism returns to the lowest 1024-byte sector of memory to find the effective address. This allows indirect addresses for

Bit 12	Bit 11	Bit 10	Address Condition
0	0	0	} Direct to lower four sectors.
0	0	1	
0	1	0	
0	1	1	
1	0	0	Direct to current sector.
1	0	1	Indirect through current sector.
1	1	0	Indexed.
1	1	1	Indirect through zero sector.

Figure 6-1. Address Bit Decoding

commonly used subroutines to be accessible from all sectors.

Indirect addressing in current sector. Address *y* designates the least-significant part of the address while the program counter specifies the most-significant part. This enables the use of

16-bit indirect addresses within the current operating sector.

Indexing

The 10-bit operand address *y* may be modified by the addition of the contents of the index register. This yields a 16-bit address that allows addressing of any location in the maximum 32,768-word memory.

Location Hexadecimal	Location Decimal	Sector	Comment
0000	0000	0	Standard memory module accessible by direct address from anywhere in memory.
03FF	1023		
0900	1024	1	
07FF	2047		
0800	2048	2	
0BFF	3071		
0C00	3072	3	
0FFF	4095		
1000	4096	4	
13FF	5119		
1400	5120	5-31	
7FFF	32,767		

Figure 6-2. Memory Sector Allocation

SECTION 7 – INSTRUCTIONS

The Varian 520/i central processor is capable of decoding and executing over 50 different instructions. This extensive instruction repertoire allows the programmer to write programs that make efficient use of machine time and minimize the space required in the core memory for program storage. The instructions contribute importantly to the Varian 520/i's versatility and real-time processing capability.

As noted in Section 5, Word Formats, the Varian 520/i instructions can take three forms: single-byte, two-byte, and three-byte.

The instructions can also be categorized by function into instructions that reference memory, those that do not reference memory, and those that relate primarily to input/output operations.

A memory-reference instruction is one that is used to store or retrieve data from memory. Two bytes are required for all such instructions since a memory address is part of the information needed to carry out the instruction.

Similarly, all input/output instructions consist of two bytes since a peripheral device must be designated for the instruction to have meaning. Thus the second byte of the I/O instruction always carries a peripheral-device address.

The third category, non-memory reference, deals with data transfers and processing within the central processor. Depending on the complexity of the operation, they require one, two, or three bytes.

Each instruction requires an execution time that is a function of the complexity of the operation and the operating precision (1, 2, 3, or 4 bytes per word). Execution time is normally expressed in terms of machine cycles, with each cycle requiring 1.5 microseconds.

On the following pages are listed all of the Varian 520/i instructions, their definitions, and their execution times. Also shown are the mnemonics used when writing a symbolic program for processing by the Varian 520/i Assembler (see Sections 16 and 17) and the equivalent binary "machine" instructions recognized by the computer. The latter are given in hexadecimal notation, with each numeral representing four binary bits.

The text-page reference for each instruction refers to the detailed description of that instruction in the Appendix. Also in the Appendix is an alphabetical index of all the instructions, based on their mnemonics, plus a numerical index based on their hexadecimal codes.

Mnemonic	Hexadecimal Code	Addressing Mode	Definition	Execution Cycles				Text Page
				OP 1	OP 2	OP 3	OP 4	
LOD	40 --	Direct	Load the current	3	4	5	6	A-42
	50 --	Direct, current sector	accumulator	3	4	5	6	
	54 --	Indirect, current sector*	from memory	5	6	7	8	
	58 --	Indexed		3	4	5	6	
	5C --	Indirect, zero sector*		5	6	7	8	
STO	60 --	Direct	Store the current	3	4	5	6	A-42
	70 --	Direct, current sector	accumulator	3	4	5	6	
	74 --	Indirect, current sector*	in memory	5	6	7	8	
	78 --	Indexed		3	4	5	6	
	7C --	Indirect, zero sector*		5	6	7	8	
SUM	80 --	Direct	Add memory to the	3	4	5	6	A-42
	90 --	Direct, current sector	current	3	4	5	6	
	94 --	Indirect, current sector*	accumulator	5	6	7	8	
	98 --	Indexed		3	4	5	6	
	9C --	Indirect, zero sector*		5	6	7	8	
SUB	A0 --	Direct	Subtract memory from the	3	4	5	6	A-43
	B0 --	Direct, current sector	current	3	4	5	6	
	B4 --	Indirect, current sector*	from the	5	6	7	8	
	B8 --	Indexed	current	3	4	5	6	
	BC --	Indirect, zero sector*	accumulator	5	6	7	8	

*Two machine cycles are required for each additional level of indirect addressing.

Figure 7-1. Two-Byte Memory-Reference Instructions (Sheet 1)

Mnemonic	Hexadecimal Code	Addressing Mode	Definition	Execution Cycles				Text Page
				OP 1	OP 2	OP 3	OP 4	
AND	C0 --	Direct	AND	3	4	5	6	A-43
	D0 --	Direct, current sector	memory to	3	4	5	6	
	D4 --	Indirect, current sector*	the current	5	6	7	8	
	D8 --	Indexed	accumulator	3	4	5	6	
	DC --	Indirect, zero sector*		5	6	7	8	
BRU	E0 --	Direct	Branch un-	2	2	2	2	A-43
	F0 --	Direct, current sector	conditionally	2	2	2	2	
	F4 --	Indirect, current sector*		4	4	4	4	
	F8 --	Indexed		3	3	3	3	
	FC --	Indirect, zero sector*		4	4	4	4	
*Two machine cycles are required for each additional level of indirect addressing.								

Figure 7-1. Two-Byte Memory-Reference Instructions (Sheet 2)

Mnemonic	Hexadecimal Code	Definition	Execution Cycles				Text Page
			OP 1	OP 2	OP 3	OP 4	
HLT	00	Halt with program counter at next location	1	1	1	1	A-46
CST	01	Change environmental status	1	1	1	1	A-53
SOV	02	Skip on current accumulator overflow and reset overflow indicator	1	1	1	1	A-49
SOVN	03	Skip on no current accumulator overflow and reset overflow indicator	1	1	1	1	A-49
SOD	04	Skip if current accumulator is odd	1	1	1	1	A-50
SODN	05	Skip if current accumulator is not odd	1	1	1	1	A-50
SAN	06	Skip if current accumulator is negative	1	1	1	1	A-48
SANN	07	Skip if current accumulator is positive	1	1	1	1	A-48
SAZ	08	Skip if current accumulator is zero	1	2	2	2	A-48
SAZN	09	Skip if current accumulator is not zero	1	2	2	2	A-48
SXZ	0A	Skip if current index register is zero	2	2	2	2	A-49
SXZN	0B	Skip if current index register is not zero	2	2	2	2	A-49
CLA	0C	Clear: current accumulator to the set precision	1	2	2	2	A-44

Figure 7-2. One-Byte Non-Memory-Reference Instructions (Sheet 1)

Mnemonic	Hexadecimal Code	Definition	Execution Cycles				Text Page
			OP 1	OP 2	OP 3	OP 4	
CMA	0D	Complement current accumulator to the set precision	1	2	2	2	A-44
IAR	0E	Increment current accumulator to the set precision	1	2	2	2	A-44
CIA	0F	Complement and increment current accumulator to the set precision	1	2	2	2	A-44
SP1	10	Set current operand precision to 1 byte	1	1	1	1	A-46
SP2	11	Set current operand precision to 2 bytes	1	1	1	1	A-46
SP3	12	Set current operand precision to 3 bytes	1	1	1	1	A-46
SP4	13	Set current operand precision to 4 bytes	1	1	1	1	A-46
SOF	14	Set current overflow indicator	1	1	1	1	A-47
ROF	15	Reset current overflow indicator	1	1	1	1	A-47
NOP	16	No operation	1	1	1	1	A-47
IXO	17	Increment current index register by the set precision	2	2	2	2	A-45

Figure 7-2. One-Byte Non-Memory-Reference Instructions (Sheet 2)

Mnemonic	Hexadecimal Code	Definition	Execution Cycles				Text Page
			OP 1	OP 2	OP 3	OP 4	
SR1	18	Shift current accumulator right one bit to the set precision	1	2	2	2	A-51
SR8	19	Shift current accumulator right eight bits to the set precision	1	2	2	2	A-51
RR1	1A	Rotate current accumulator right one bit to the set precision	2	2	2	3	A-51
RR8	1B	Rotate current accumulator right eight bits to the set precision	1	2	2	2	A-51
SL1	1C	Shift current accumulator left one bit to the set precision	1	2	2	2	A-52
SL8	1D	Shift current accumulator left eight bits to the set precision	1	2	2	2	A-52
RL1	1E	Rotate current accumulator left one bit to the set precision	2	2	2	3	A-52
RL8	1F	Rotate current accumulator left eight bits to the set precision	1	2	2	2	A-52

Figure 7-2. One-Byte Non-Memory-Reference Instructions (Sheet 3)

Mnemonic	Hexadecimal Code	Definition	Text Page
Z	0=s, d	Nonexistent zeros register	A-54
A	1=s, d	Current accumulator most-significant 16 bits	A-54
C	2=s, d	Current accumulator least-significant 16 bits	A-54
X	3=s, d	Current index register	A-54
P	4=s, d	Current program counter	A-54
B	5=s, d	Noncurrent accumulator most-significant 16 bits	A-54
D	6=s, d	Noncurrent accumulator least-significant 16 bits	A-54
Y	7=s, d	Noncurrent index register	A-54
Q	8=s, d	Noncurrent program counter	A-54
F	9=s, d	Peripheral adapter input/output register	A-54

Figure 7-3. Two-Byte Non-Memory-Reference Instructions (Sheet 1)

Mnemonic	Hexadecimal Code	Definition	Execution Cycles				Text Page
			OP 1	OP 2	OP 3	OP 4	
T	20 s d	Transfer source register to destination register	3	3	3	3	A-54
C	21 s d	Complement source register and transfer to destination register	3	3	3	3	A-55
I	22 s d	Increment source register and transfer to destination register	3	3	3	3	A-55
D	23 s d	Decrement source register and transfer to destination register	3	3	3	3	A-55
M	24 s d	AND source register into destination register	3	3	3	3	A-55
O	25 s d	OR source register into destination register	3	3	3	3	A-56
E	26 s d	Exclusive OR source register into destination register	3	3	3	3	A-56
A	27 s d	Add source register to destination register	3	3	3	3	A-56

Figure 7-3. Two-Byte Non-Memory-Reference Instructions (Sheet 2)

Mnemonic	Hexadecimal Code	Definition	Execution Cycles				Text Page
			OP 1	OP 2	OP 3	OP 4	
MIN	3000	Modify interrupts according to current accumulator	2	2	2	2	A-59
CIS	3100	Copy interrupt status into current accumulator	2	2	2	2	A-60
CIM	3120	Copy interrupt mask into current accumulator	2	2	2	2	A-61
SIE	3200	Skip if in interruptable environment	3	3	3	3	A-60
SSH	3260	Skip on Power-fail interrupt in halt mode	3	3	3	3	A-60
SS1	3220	Skip if SENSE switch 1 is off	3	3	3	3	A-59
SS2	3240	Skip if SENSE switch 2 is off	3	3	3	3	A-59
SS3	3280	Skip if SENSE switch 3 is off	3	3	3	3	A-60
IBD	3300	Interrupt system disable	2	2	2	2	A-61
IBE	3320	Interrupt system enable	2	2	2	2	A-61

Figure 7-3. Two-Byte Non-Memory-Reference Instructions (Sheet 3)

Mnemonic	Hexadecimal Code	Definition	Execution Cycles				Text Page
			OP 1	OP 2	OP 3	OP 4	
BRM	38----	Branch and mark (store) current program counter	6	6	6	6	A-65
CSQ	39----	Change environmental status and load noncurrent program counter	3	3	3	3	A-65

Figure 7-4. Three-Byte Non-Memory-Reference Instructions

Mnemonic	Hexadecimal Code	Definition	Execution Cycles				Text Page
			OP 1	OP 2	OP 3	OP 4	
BTO	30 ..	Byte transfer out of current accumulator	2	2	2	2	A-57
BTI	31 ..	Byte transfer into current accumulator	2	2	2	2	A-57
SEN	32 ..	Sense device and skip if sense true	3	3	3	3	A-57
FUN	33 ..	Transfer function out and transfer a byte out of the current accumulator	2	2	2	2	A-58

Figure 7-5. Two-Byte Input/Output Instructions

Mnemonic	Hexadecimal Code	Definition	Execution Cycles				Text Page
			OP 1	OP 2	OP 3	OP 4	
BTO 01	3001	Transfer least-significant byte or current accumulator to the teletype controller input buffer	2	2	2	2	A-63
BTI 01	3101	Transfer teletype controller buffer to the least-significant byte of the current accumulator	2	2	2	2	A-63
SEN 21	3221	Sense for teletype controller output buffer readiness and skip if ready	3	3	3	3	A-63
SEN 41	3241	Sense for teletype controller input buffer readiness and skip if ready	3	3	3	3	A-64
FUN 01	3301	Enable teletype controller write interrupt	2	2	2	2	A-63
FUN 21	3321	Reset teletype controller	2	2	2	2	A-63
FUN 41	3341	Start teletype punched-tape reader	2	2	2	2	A-63
FUN 61	3361	Enable teletype controller read interrupt	2	2	2	2	A-63
FUN C1	33C1	Read one character on the teletype punched-tape reader	2	2	2	2	A-63

Figure 7-6. Teletype Controller Reserved Instructions

Mnemonic	Hexadecimal Code	Definition	Execution Cycles			
			OP 1	OP 2	OP 3	OP 4
BTO 02	3002	Punch least-significant byte of current accumulator	2	2	2	2
BTI 02	3102	Read one tape character into least-significant byte of current accumulator	2	2	2	2
SEN 02	3202	Sense for punched-tape transfer readiness and skip if ready	3	3	3	3
FUN 22	3322	Start punched-tape reader	2	2	2	2
FUN 42	3342	Enable punched-tape interrupt	2	2	2	2
FUN A2	33A2	Stop punched-tape reader	2	2	2	2
FUN C2	33C2	Disable punched-tape interrupt	2	2	2	2

Figure 7-7. Punched-Tape System Reserved Instructions

SECTION 8 – INPUT/OUTPUT BUS

The Varian 520/i has three basic methods for transferring data to and from external devices:

1. An eight-bit, program-controlled data transfer to and from the accumulator, via the standard I/O bus.
2. A 16-bit, program-controlled data transfer to and from the accumulator via a 16-bit I/O channel (Section 27).
3. An eight-bit direct memory access (DMA) channel to and from the computer memory (Section 10).

The standard Varian 520/i is equipped with the eight-bit I/O channel and the ability to accommodate DMA. The 16-bit data-transfer channel is optional.

I/O Bus

The basic Varian 520/i machine provides five types of I/O operations, all of them associated with the standard I/O bus.

1. **Sense** – The status of a selected peripheral controller sense line is interrogated by the computer under program control.
2. **Function control** – A control code is transferred under program control to a peripheral controller.

3. **Byte transfer in** – A single byte of data is transferred under program control from a peripheral controller to the least-significant eight bits of the current accumulator.

4. **Byte transfer out** – A single byte of data is transferred under program control to a peripheral controller from the least-significant eight bits of the current accumulator.

5. **Interrupt** – A peripheral controller transmits an interrupt request to the computer to initiate special program subroutines.

The standard I/O bus consists of a data bus, address bus, interrupt bus and six control lines.

The data bus is an eight-bit, parallel, bidirectional I/O channel. It is used to transmit data from the computer to peripheral devices. In turn the bus is used by these devices to transmit data to the computer. A total of 10 drivers and 10 receivers may be connected to each line. Data bus mnemonics are ED00 to ED07. A typical data bus line is shown in Figure 8-1.

The address bus is an eight-bit output bus. It is used to transmit device address and order code to the peripheral device. Five bits determine device address and three bits specify up to eight control functions or

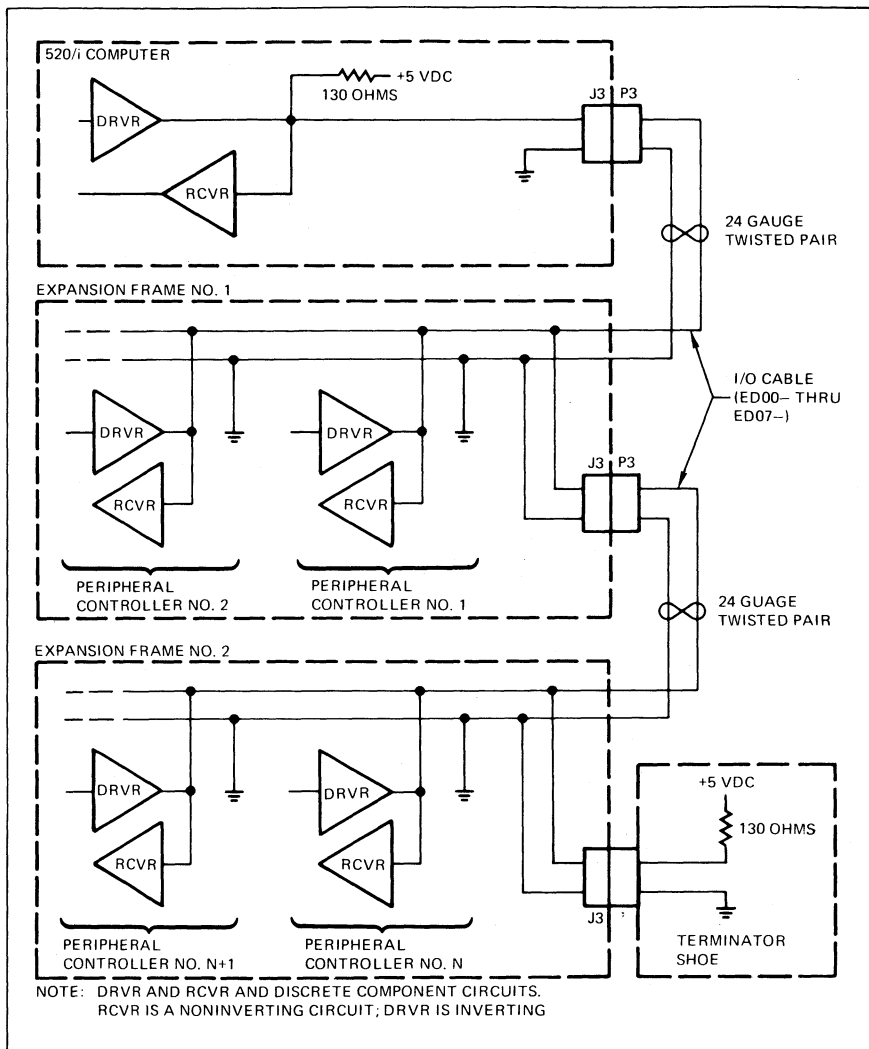


Figure 8-1. Typical Varian 520/i Data Bus Line

sense inquiries. A total of 10 receivers may be connected to each output. Address bus mnemonics are EA00 to EA07. A typical address bus line is shown in Figure 8-2.

The interrupt bus consists of 11 interrupt lines. Highest priority is assigned by hardware to three lines (EI00, EI01, and EI02-). The eight remaining lines are assigned priority by software. Interrupt receivers may be driven by up to 10 drivers. Interrupt bus mnemonics are EI00- to EI11. Typical interrupt lines are shown in Figure 8-12 and the interrupt system is explained in detail in Section 9.

Control lines are provided for each of six control signals. Four of these, FUNS-, SENS-, BTIS-, and BTOS-, are generated during computer-initiated I/O functions. They are mutually exclusive and determine which of four operations is in process. Internal gating provides a nominal signal duration of 500 nanoseconds, for the FUNS-, BTIS-, BTOS- signals and 1250 for the SENS- signal. The four signals are defined as follows:

FUNS-	Functional control
SENS-	Sense external device.
BTIS-	Byte transfer in.
BTOS-	Byte transfer out.

Control signal SERPS- is an affirmative sense response. The receiver for this signal may be driven by up to 10 drivers.

Control signal RSTS- is true during the period in which the console RESET switch is pressed. This signal may drive up to 10 receivers.

Control signal CRUN- indicates that the Varian 520/i is in the RUN mode.

Typical control lines are shown in Figures 8-2 and 8-3.

Line Drivers and Receivers

The line drivers and receivers used on the I/O bus enable a wired-OR, negative-logic junction. Negative logic in this sense refers to a signal with a 1 value near ground and a zero value in excess of 2.4 volts. Consequently, all signals on the I/O bus are complements of signals used in the Varian 520/i and the peripheral equipment.

I/O Instructions

The basic two-byte I/O instructions are:

BTI, BTO	Byte transfer in and out of lower accumulator.
SEN	Sense I/O device condition.
FUN	Command I/O device to perform a particular function.

The second byte of the I/O instructions contain an associated order code and device address which is transmitted to the external devices on the eight-bit address bus.

The format for the second byte is:

7	6	5	4	3	2	1	0
ORDER				DEVICE			

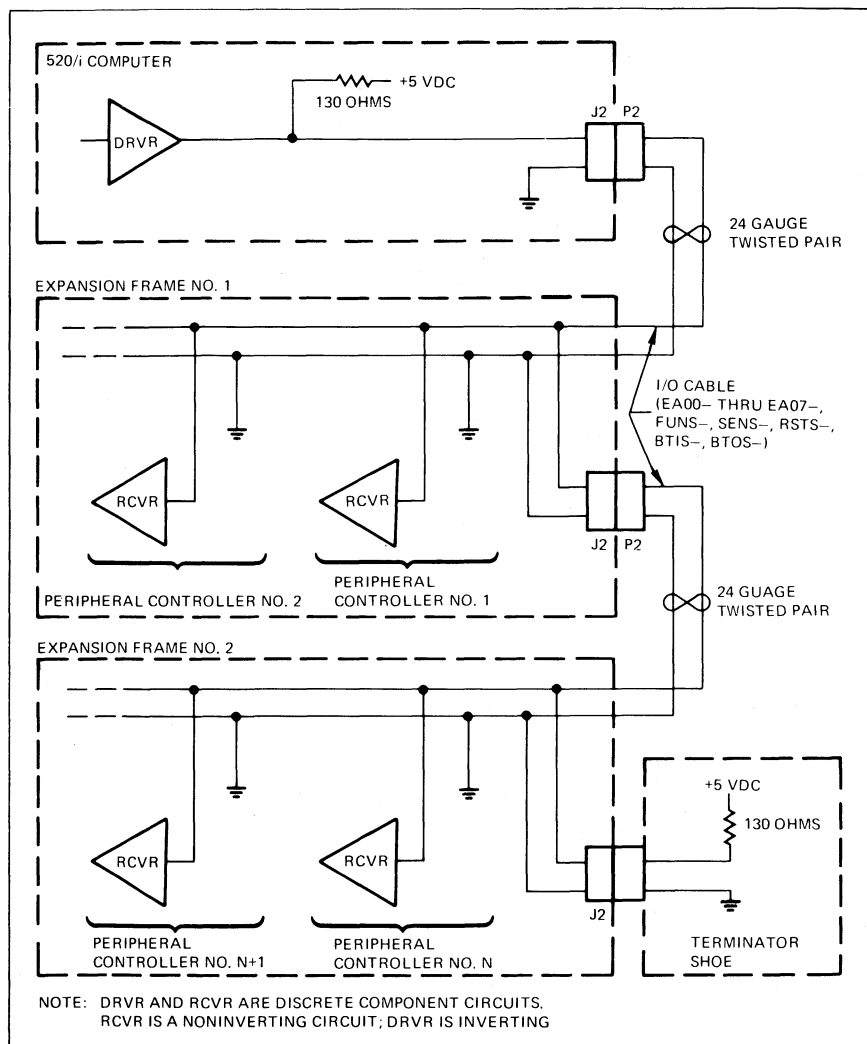


Figure 8-2. Typical Control or Address Line from the Varian 520/i

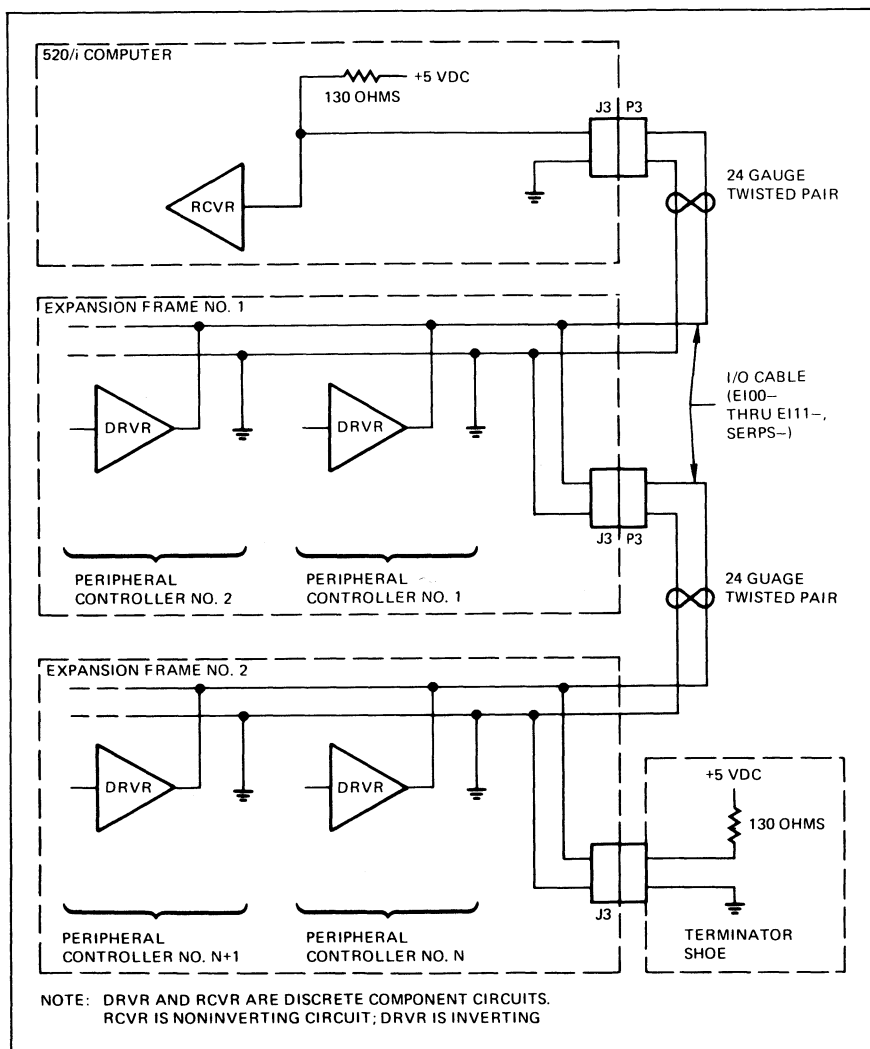


Figure 8-3. Typical Control Line to the Varian 620/i

input/output bus

Each device uses a unique device address, bits 0-4, with up to 32 device addresses possible. Bits 5-7 identify the function to be performed or in some other way qualify the I/O command. Assigned addresses for standard peripherals are given in the table on page A-39.

The devices attached to the I/O bus operate in the sense-loop mode or in the transfer-on-interrupt mode, or both. In the sense-loop mode, the computer commands the device to perform a function, senses until the device is ready to transfer data, then executes a byte-transfer instruction.

In the transfer-on-interrupt mode, the device is alerted to perform a function, the central processor enables its interrupt line and proceeds in its program until the device requests a byte transfer by activating an interrupt line. The program discontinues its current process and branches to a sub-routine for input or output for the interrupting device.

The following is a typical sense-loop transfer program. A device is alerted, data is transferred, and the central processor tests for readiness for the next transfer. The maximum transfer rate in this example is 47,600 bytes per second.

		Cycles (Min.)
SPI	(Set accumulator precision to one byte)	1
FUN DA	(Alert device)	2
LOD X	(Load accumulator, indexed)	3

		Cycles (Min.)
SEN DA	(Sense device readiness)	3
BRU-2	(Sense again if sense false)	2
BTO	(Output one byte if sense true)	2
IXO	(Increment index register)	2
SXZ	(Sense for index register = zero)	2
BRU	(Branch to LOD X for next byte)	2
		14 cycles mini- mum
EXIT	(Exit if index register zero)	

14 cycles x 1.5 microseconds = 21.0 microseconds.

Typical Controller Circuits

The I/O bus is designed to provide efficient and flexible communications with a variety of peripheral equipment. A number of external devices can be controlled with a minimum of control logic.

The examples that follow depict typical situations encountered in the control of peripheral devices. It is assumed that the device controllers interface with the eight-bit I/O bus using drivers and receivers compatible with those found in the Varian 520/i.

All timing requirements assume a 30-foot cable is to be driven. If shorter cables are used, minimum times may be increased by 3.3 nanoseconds per foot for each foot of cable length less than 30 feet.

Sense status of external device. The status of an external device is interrogated by the use of a SEN instruction. The least-significant part of the address portion of the instruction determines which of 32 devices is to be interrogated. One of the eight sense functions is selected by coding the most-significant part of the address portion of the instruction.

The logic for the sense function for a typical controller is shown in Figure 8-4. The address portion of the instruction precedes the SENS- signal by at least 125 nanoseconds at the output of the receivers. The SENS-signal has a minimum duration of 1150 nanoseconds and a maximum duration of 1350 nanoseconds at the output of the receiver.

The sense response (SERPS-) must therefore be applied to the NAND gate feeding the driver within 750 nanoseconds of the receipt of SENS-signal from the receiver. The sense response must have a minimum duration of 250 nanoseconds with a maximum duration of 2 microsecond. Timing requirements are shown in Figure 8-6.

The circuit of Figure 8-4 satisfies these conditions by using SENS+ to strobe the sense response. This provides a minimum delay and controls the sense response duration. Figure 8-4 also shows the decoding of a typical address. This function, ADDR, is used in other illustrations.

External control function. Control signals are applied to external devices using the FUN instruction. The least significant part of the address portion of the instruction determines which of 32 devices is to receive the control signal. One of eight control functions can be selected by coding the most significant part of the address portion of the instruction.

The logic for a typical control function is shown in Figure 8-5. The address portion of the instruction precedes the FUNS- signal by at least 125 nanoseconds at the output of the receivers. The FUNS- signal has a minimum duration of 400 nanoseconds and a maximum duration of 600 nanoseconds.

The circuit of Figure 8-5 uses the FUNS-signal to control the duration of the control signal. If a greater duration is required, the FUNS- signal may be used to set a flip-flop. Timing requirements are shown in Figure 8-7.

Data transfer output operations. Data are transferred from the computer to a peripheral device by means of a BTO instruction. Data are transmitted along data bus lines ED00- to ED07-. The operation is recognized as an output transfer by the presence of the BTOS-signal. The address portion of the instruction determines the peripheral device to receive the data. The most-significant three bits of the address portion of the instruction may be used to distinguish several types of output bytes to a single device.

The logic to implement a data transfer for a typical situation is shown in Figure 8-8. The address portion of the instruction precedes

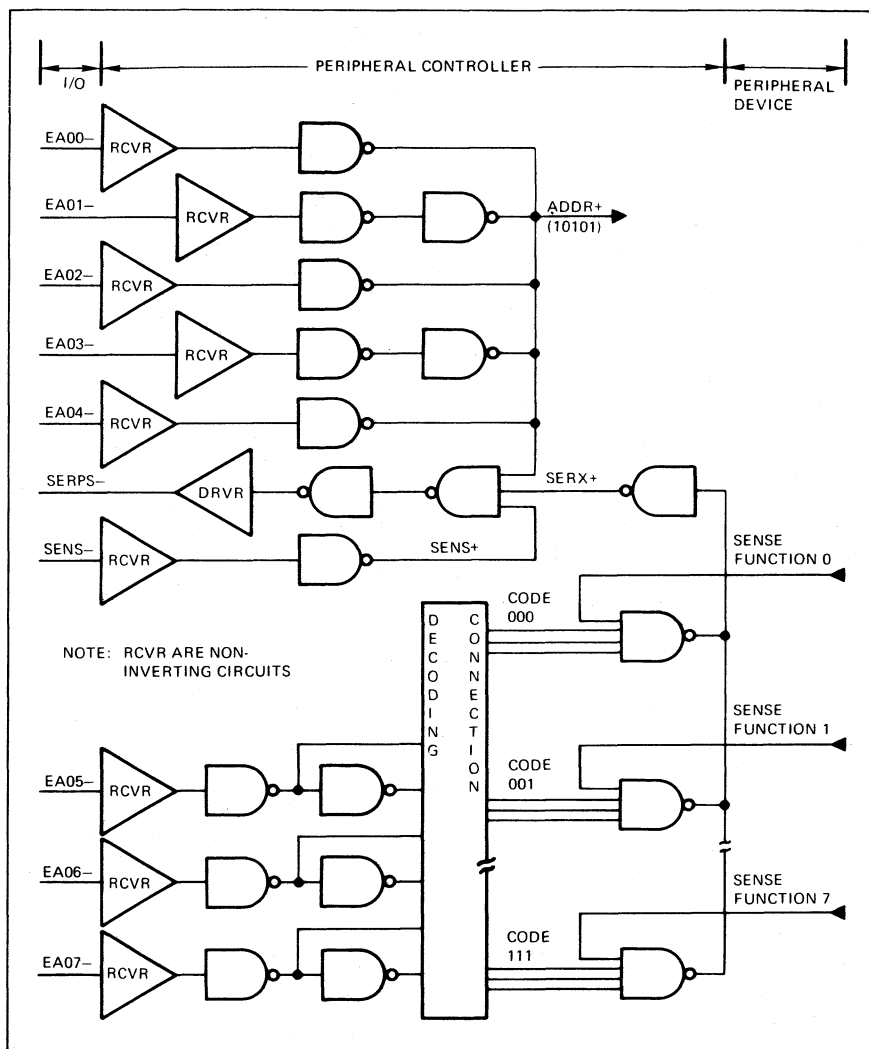


Figure 8-4. Typical Varian 520/i Sense Logic

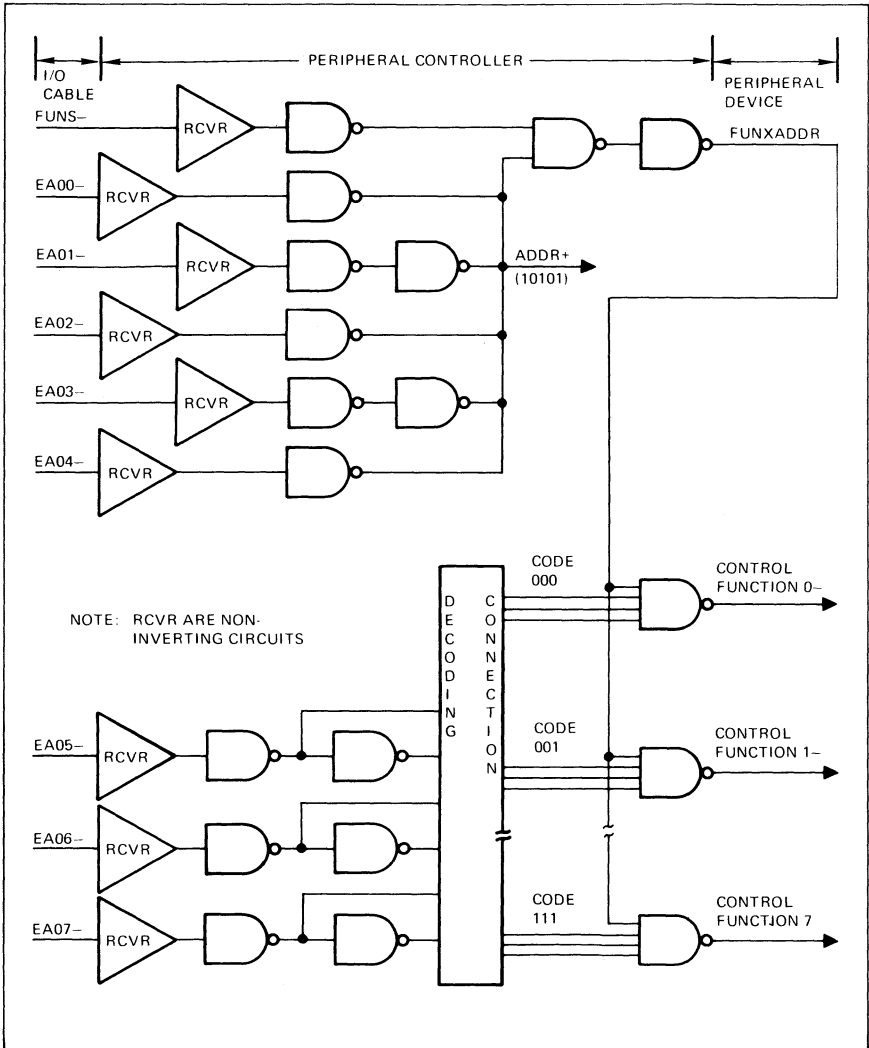


Figure 8-5. Typical Varian 520/i Function Control Logic

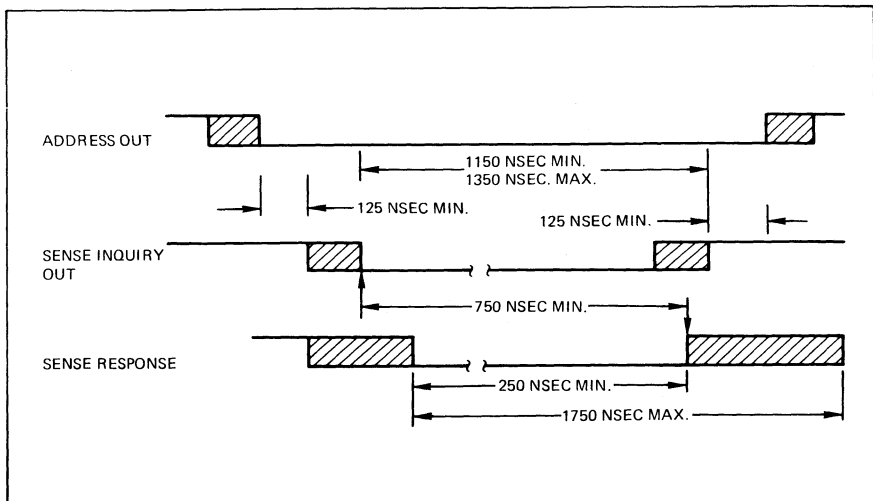


Figure 8-6. External Sense Timing

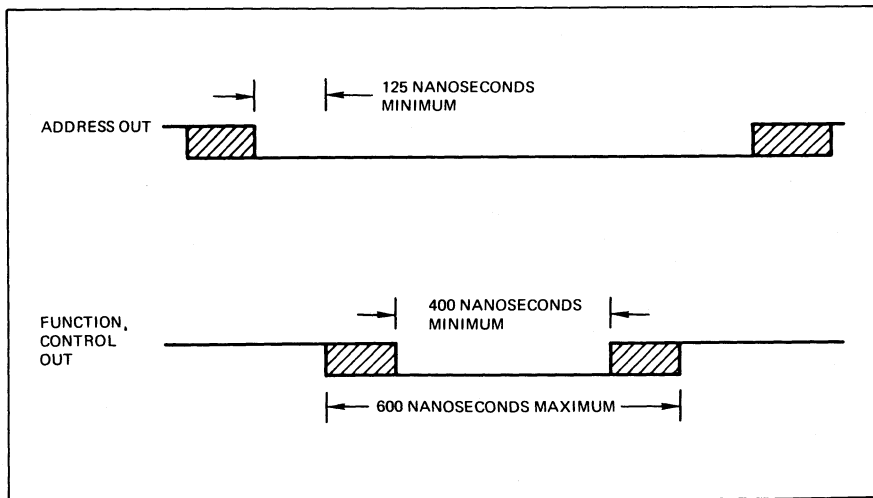


Figure 8-7. External Function Control Timing

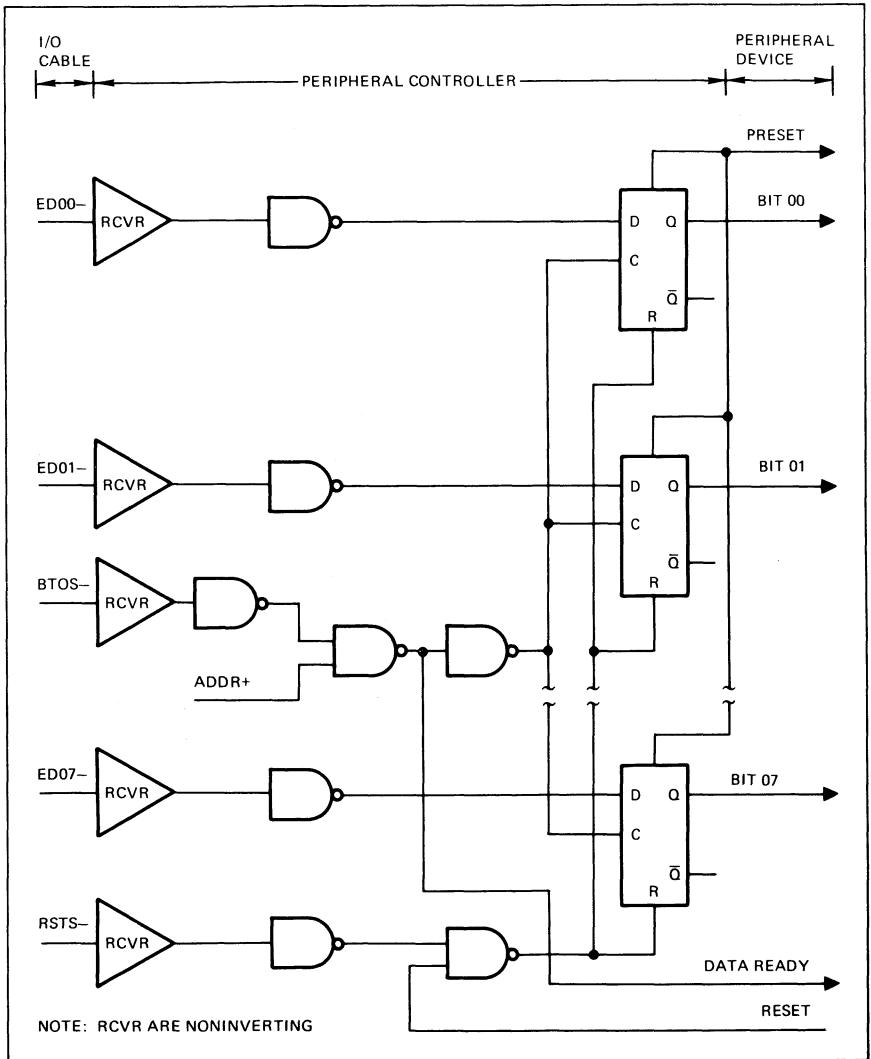


Figure 8-8. Typical Varian 520/i Data Transfer Output Logic

the BOTS-signal by at least 125 nanoseconds at the output of the receivers. Data is present at the receivers at least 140 nanoseconds before BTOS-. The BTOS-signal is at least 400 nanoseconds but not more than 600 nanoseconds in duration. Data is present at the receivers for a minimum of 900 nanoseconds. These timing conditions are shown in Figure 8-10.

The circuit of Figure 8-8 applies the data directly into the inputs of "type D" flip-flops. BTOS- is gated with the address and is used as a clock. In this configuration, the flip-flops present the new data within 240 nanoseconds of the receipt of BTOS- (assuming TTL flip-flops). The complement of the clock may be used as a data ready signal.

A typical application of the RSTS- signal is shown in Figure 8-10. The signal can be "ORed" with a signal from the peripheral device. This enables the register to be reset by the Varian 520/i or by the peripheral device.

Data transfer input operations. Data is transferred from a peripheral device to the computer by means of a BTI instruction. Data is transmitted along data bus lines

ED00- to ED07-. The operation is recognized as an input transfer by the presence of the BTIS- signal. The address portion of the instruction determines the peripheral device to send the data. The most-significant three bits of the address portion of the instruction may be used to distinguish several types of input bytes from a single device.

The logic to implement a data transfer for a typical situation is shown in Figure 8-9. The address portion of the instruction precedes the BTIS- signal by at least 125 nanoseconds at the output of the receivers. The BTIS-signal is at least 400 nanoseconds but not more than 600 nanoseconds in duration. Data must be present at the drivers until the end of the BTIS- signal. The leading edge of data at the drivers should occur no later than 110 nanoseconds after the leading edge of BTIS-. Timing requirements are shown in Figure 8-11.

The logic of Figure 8-9 presumes that data is ready in the peripheral devices before the BTI instruction. Either the computer was informed that data is ready by an affirmative sense response or an interrupt, or the computer readied the data by means of a function control instruction. The timing requirements are satisfied by using the BTIS+ signal as a strobe.

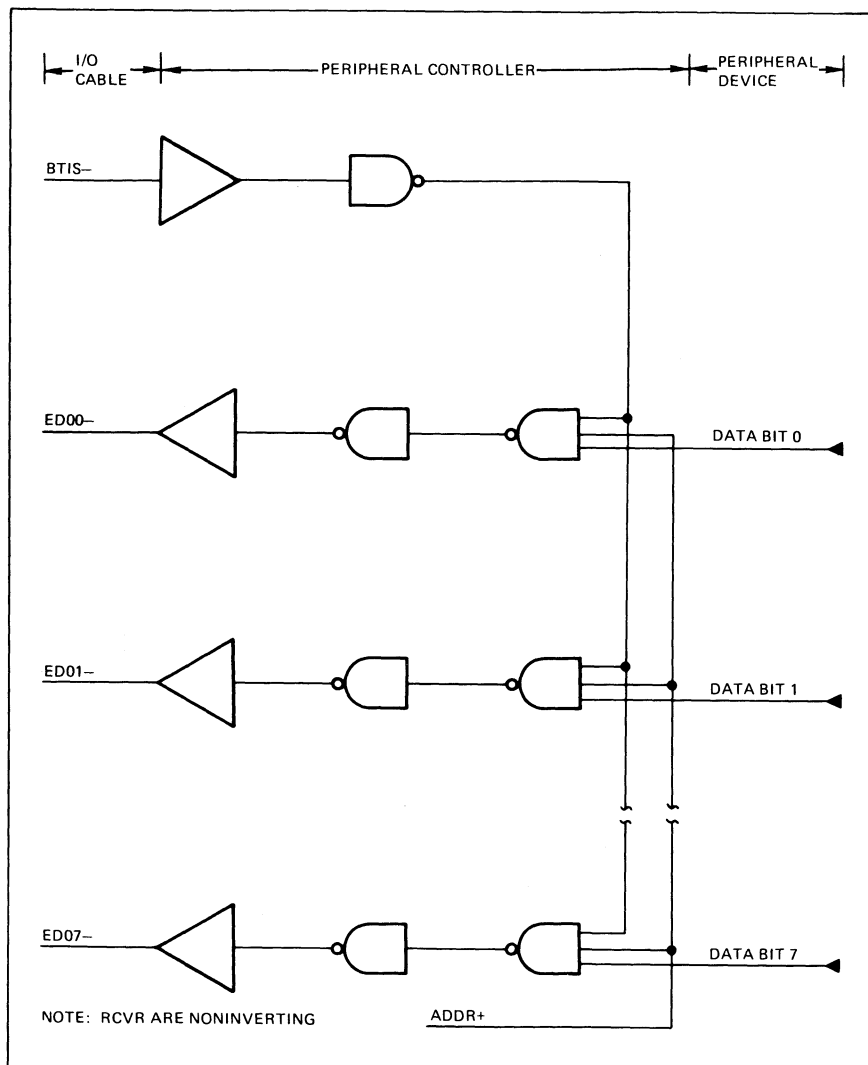


Figure 8-9. Typical Data Transfer Input Logic

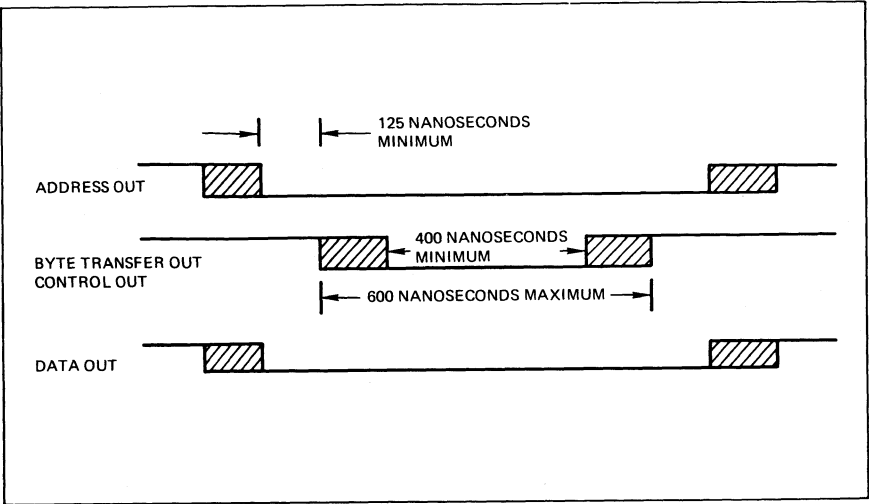


Figure 8-10. Data Output Timing Requirements

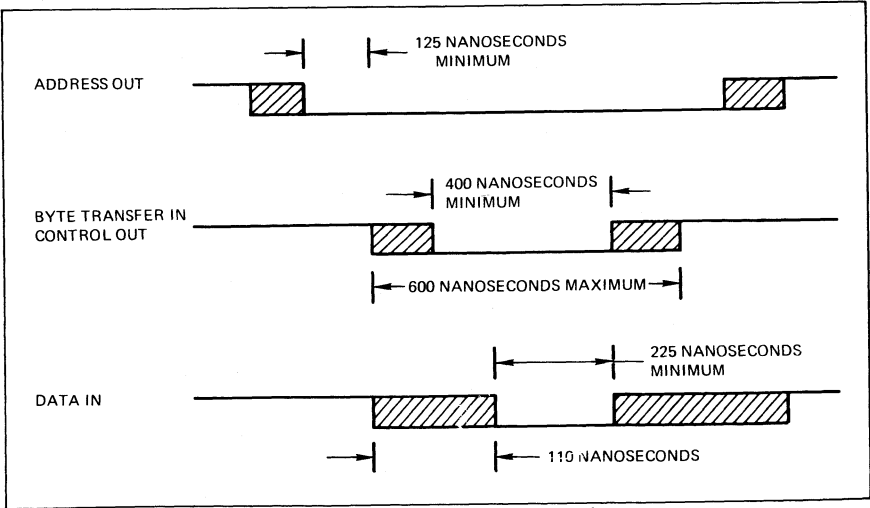


Figure 8-11. Data Input Timing Requirements

SECTION 9 – PRIORITY INTERRUPT SYSTEM

The priority interrupt system relieves the main program of the need for repeated status checks by allowing the I/O devices to cause a program interrupt. The computer must be in the interruptable environment to receive interrupts (other than a parity or a power fail/restart interrupt).

As shown in Figure 9-1, there are eleven interrupt lines (00 to 10) which are combined into four priority levels (0 to 3). Interrupt lines 3 to 10 all have the same hardware priority level and make up interrupt number 3. The programmer assigns priorities within this group by copying these eight interrupt lines into the accumulator and determining the interrupting device.

Figure 9-2 details the interrupt system and its component parts.

The I/O Cable contains the eleven interrupt lines available to the I/O devices. (There are an additional two internal interrupts for the computer options, power fail/restart and parity.)

The level required on these interrupt lines are of the DC type. The interrupt line from an I/O device must remain in the active state until the I/O device is acknowledged by the computer by means of an I/O command. At such time the interrupt level is made inactive by the I/O device.

DC Latches are used to store the status of the interrupt lines until sampled. The inter-

rupt lines are sampled at a time corresponding to the beginning of fetching the program counter value for use as the memory address for the next instruction.

Once during each timing cycle (1.5 microseconds) the DC latches are reset to clear the latches of any interrupts which have been acknowledged by the computer and made inactive by the I/O device.

Following the reset of the latches they are again set with the value of their corresponding interrupt line. The latches are set at such a time that the priority level determinator and address generator have sufficient time to settle before the computer samples them for their status. This prevents any problems due to I/O devices operating asynchronously to the computer.

Interrupt Masks are used to enable or disable individual interrupt lines. The masks are selectively enabled or disabled by a Modify Interrupt instruction. The masks, when enabled, allow the corresponding interrupt to be input to the interrupt determinator. When disabled, the interrupt mask inhibits the acknowledgement of the interrupt on the corresponding interrupt line.

Interrupt Determinator establishes the relative priority of the interrupt lines. The order of priority is such that the power fail/restart line has highest priority, next

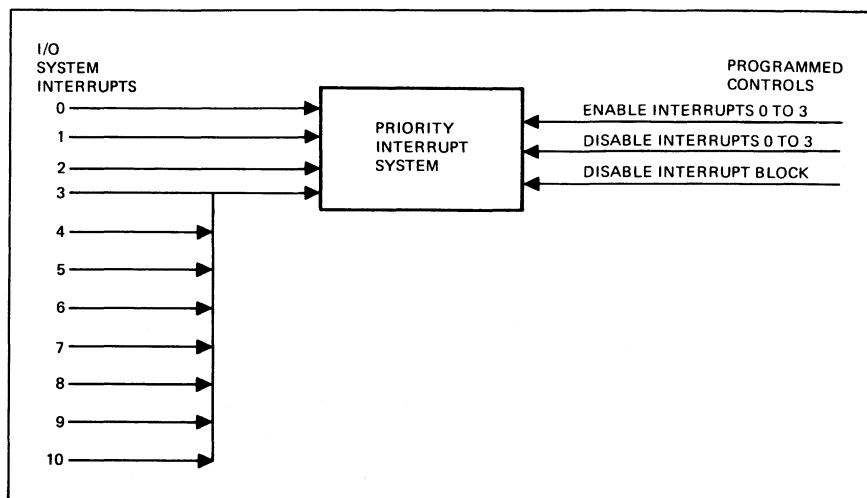


Figure 9-1. Interrupt Configuration

highest priority is the parity line and following these in priority are the I/O interrupt lines 0-3, of which line 3 has the lowest priority. Priority among the eight low-priority interrupts that share line 3 is assigned by software.

If two or more interrupts occur at once, the line with the highest priority is given precedence. An interrupt line is allowed to request an interrupt only if it has not been masked off and a higher priority line is not requesting.

Interrupt Address Generator generates the memory address of the interrupt line requesting the interrupt. The address corresponding to each interrupt line is given below.

	Address Priority (Hexadecimal)	
Power Fail Interrupt (optional)	1	0000
Parity Error (optional)	2	0004
Interrupt Line 0	3	0008
Interrupt Line 1	4	000C
Interrupt Line 2	5	0010
Interrupt Line 3-10	6	0014

Interrupt Block Enable determines whether the interrupt system, as a block, is enabled. The system is enabled only after the

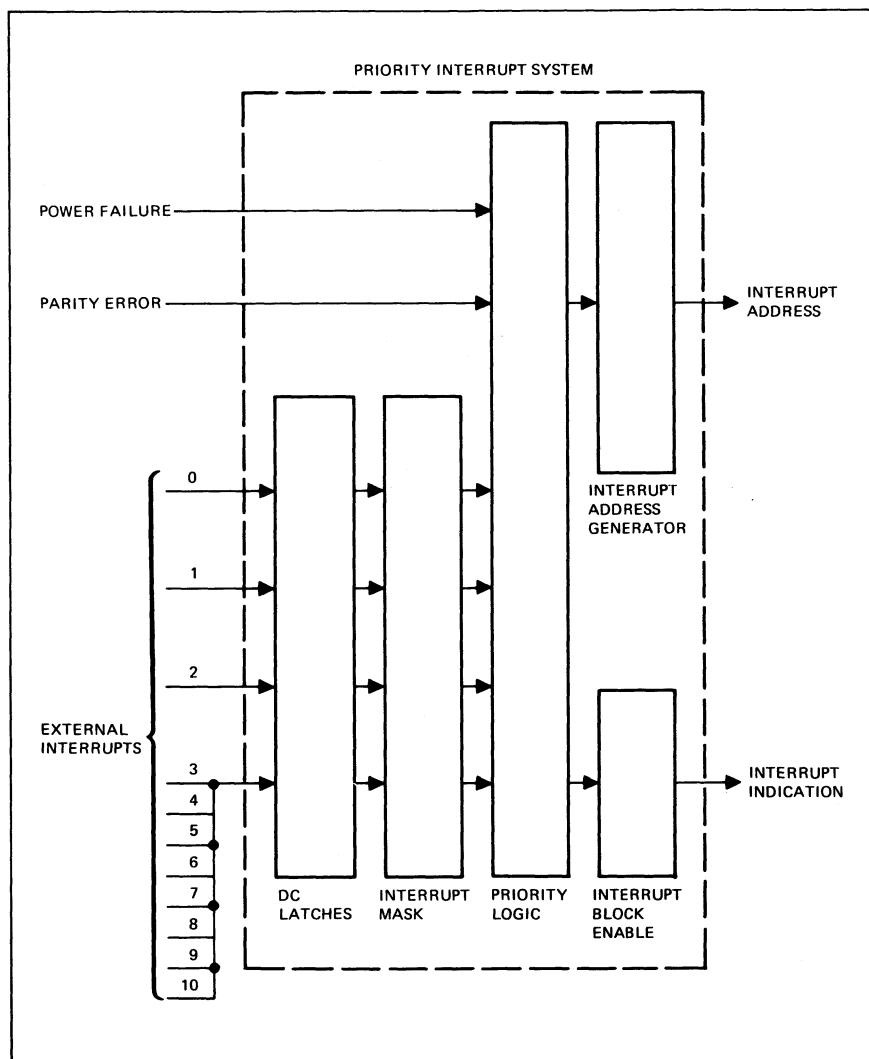


Figure 9-2. Varian 520/i Priority Interrupt System

priority interrupt system

computer executes the IBE (Interrupt Block Enable) instruction. With the block disabled, all interrupts except the power fail/restart interrupt are inhibited from interrupting the computer. The power fail/restart interrupts the computer regardless of block enable or environment.

The interrupt system, as a block, is reset (thus requiring an IBE instruction when it is desired to enable it again) under the following conditions:

1. When power is first applied.
2. When the RESET switch on the front panel is activated.
3. When the computer executes a DBE (Disable Block Interrupt) instruction.
4. When the computer is interrupted.

Interrupt System Instructions

The following program instructions apply to the interrupt system:

IBE Interrupt Block Enable

The interrupt system, as a block, is enabled by this instruction. The interrupt system is normally inactive and all interrupts, except power fail/restart, are inhibited from interrupting the computer. Also when an interrupt occurs the entire block of interrupts is disabled; so at the end of the interrupt subroutine an IBE instruction must be executed in order for the system to be receptive to any other interrupts.

DBE Interrupt Block Disable

This instruction disables the interrupt system. No interrupts, other than power fail/restart, will interrupt the computer.

CIS Copy Interrupt Status

The eight interrupt lines which share priority interrupt number 3 are copied into the least significant bits of the current accumulator.

CIM Copy Interrupt Mask

The four interrupt mask lines are copied into the least significant bits of the current accumulator.

This instruction would be used in a program where it is desirable to interrupt out of an interrupt service subroutine by a higher priority interrupt.

One requirement of allowing a higher priority interrupt during an interrupt subroutine is the determination of the interrupt mask configuration upon entering the subroutine. The knowledge of the correct mask is necessary so that at the completion of processing the interrupt, only the interrupt masks that were on when entering this subroutine are re-enabled. Without this instruction, all interrupt masks are enabled when exiting from the current interrupt service subroutine and are thus susceptible to lower priority interrupts.

MIN Modify Interrupts

The interrupts are selectively enabled or disabled by this instruction. The contents of the least significant bits of the

accumulator are used to modify the enable/disable status of the 4 interrupt lines. Bits 00-03 of the accumulator, if set to 1's, enables interrupts 0-3. Bits 04-07 of the accumulator, if set to 1's, disables interrupts 0-3.

If the "enable" bit in the accumulator is a 0 and the "disable" bit is a 0, the present status is left unchanged. If both enable and disable bits are 1's, the interrupt enable/disable status is complemented.

Interrupt Subroutine Instructions

The instruction in the interrupt location must be one of the following:

Branch and Mark (BRM),
Change Status (CST),
Change Status and Load Q (CSQ),
or Branch Unconditionally (BRU)

For Power Fail and Parity Interrupts the only acceptable instructions are BRM or BRU.

Power Fail and Parity Interrupts must be acted on immediately and therefore these interrupts are not sensitive to the machine environment, i.e., they are recognized even if the computer is in the non-interruptible environment. For this reason, they must be

handled by software, with a BRM or a BRU, to branch the program to an appropriate subroutine.

Logic Levels on Interrupt Lines

The line drivers and receivers connected to the I/O bus have a "wired-OR" connection and therefore negative logic is used. A signal is true when its value is near ground and false when in excess of 2.4 volts. To cause an interrupt, an input line must be brought to ground. Since the complement of the normal Varian 520/i logic is used, the mnemonic designations of the eleven interrupt signals are EI00- to EI10-, with EI00- having the highest priority.

Figure 9-3 shows typical interrupt circuits. Interrupt A is the usual connection with the interrupt driver connected directly to the interrupt source.

A NAND gate used at the input to the driver provides an opportunity to logically OR two interrupt sources as shown at B. Interrupt C is the logical AND of two interrupt sources. The interrupt shown at D is a logical OR of two interrupt sources from two separate controllers.

Interrupts may be raised at any time but must persist until reset by a response from the computer in the form of an I/O command.

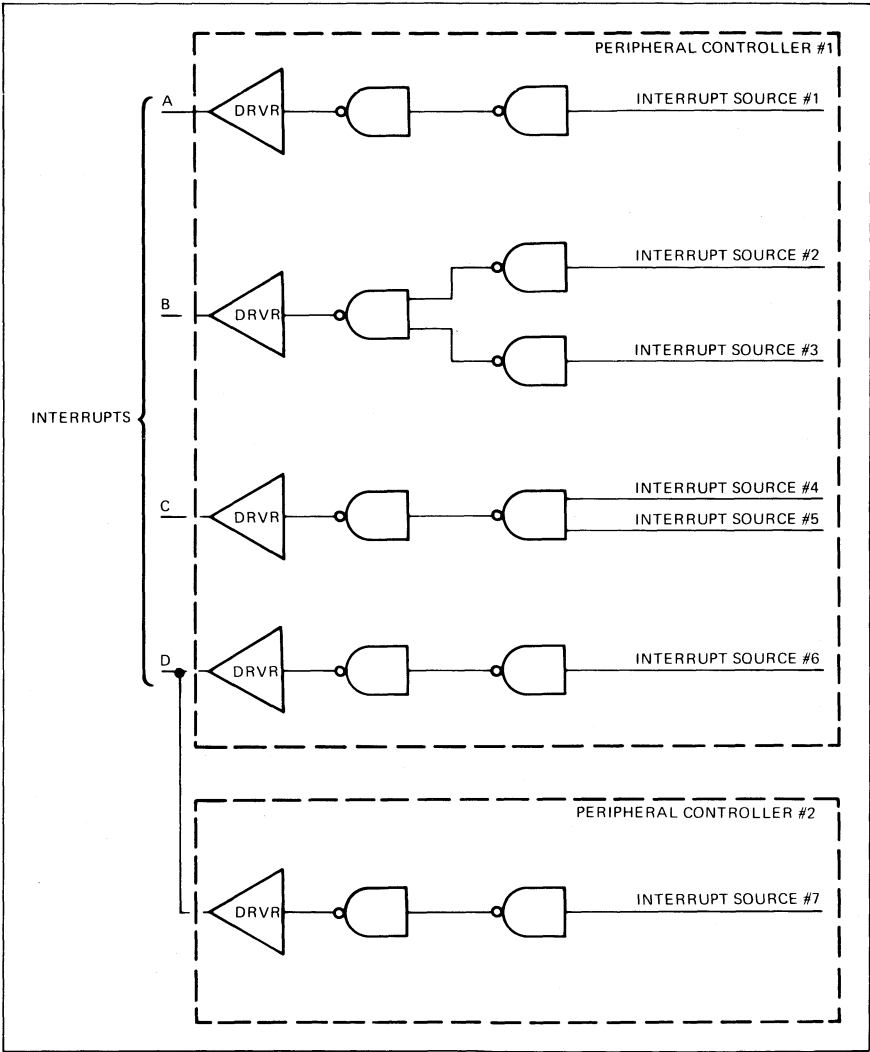


Figure 9-3. Typical Interrupt Circuits

SECTION 10 – DIRECT MEMORY ACCESS

The Varian 520/i computer has a built in Direct-Memory-Access (DMA) capability. The DMA capability enables external equipment, such as tape recorders, to interrupt the CPU and initiate a transfer to or from memory without the need of a stored instruction and without disturbing the operational registers. At the conclusion of the data transfer, the CPU program continues as if uninterrupted.

The direct connection to memory that makes this possible is known as the DMA port, and the various ways of implementing the direct transfer of data are known as the DMA options.

Since DMA options permit data transfers to and from external equipment at rates up to 666,000 bytes per second, it is ideally suited for high-speed data transfers by magnetic tape or disc units or from data sampling units.

The DMA transfers achieve these high rates by stealing cycles from the CPU. The CPU is simply inhibited from operating while the transfer is taking place. Other features of DMA transfers include:

- The external mechanism operates asynchronously, with feedback signals.
- The external mechanism provides the memory addresses.

- The external mechanism can use consecutive memory cycles for a high data-burst rate.

Data Rates

The DMA port can operate in one of two modes, depending on the data transfer rate of the external mechanism.

For external equipment operating at less than 93,000 bytes per second, a DMA request sent to the Varian 520/i is acknowledged at the completion of the instruction in process. The external data word is then stored in memory or an accessed word from memory is transferred to the external device. The maximum access rate is determined by the instruction with the longest execution time. For most programs, the worst-case instruction execution time is about 9 microseconds, giving the following maximum access rates for this first mode:

One external device	93,000 bytes per second
Two devices operating together	81,500 bytes per second
Three devices operating together	73,000 bytes per second
Four devices operating together	65,500 bytes per second

direct memory access

For external equipment operating at greater than 93,000 bytes per second, an I/O command from the Varian 520/i is sent to the requesting external device before data transfer is started. The I/O command sustains the DMA request signal throughout the data transfer. At the completion of the data transfer, the DMA request signal is made inactive, allowing the interrupted program to continue. In this way, every memory cycle during the data transfer is reserved for DMA, giving the following maximum access rates for the second mode:

One external device	660,000 bytes per second
Two devices operating together	333,000 bytes per second
Three devices operating together	222,000 bytes per second
Four devices operating together	167,000 bytes per second.

DMA Options

There are three optional ways in which the user can implement the DMA port:

1. The user can build his own DMA requesting logic on his own logic board and insert it into the vacant DMA slot in the computer. Varian Data Machines provides information, in this case, as to the capabilities, restrictions and limitations of connecting directly to the internal computer bus structure.
2. The user can purchase from Varian Data Machines a general-purpose socket board that plugs into the vacant DMA slot. Varian Data Machines then provides the

information necessary to build up the logic required.

3. The user can purchase a Varian 520/i-12 DMA Port Line Drivers/Receivers circuit board. Varian Data Machines then provides information as to how to connect the external mechanisms to the drivers and receivers on this board.

In all three of these options, the external mechanisms using the DMA port must provide address, data and control signals compatible with the Varian 520/i logic.

Internal DMA Logic

Figure 10-1 details the DMA logic within the Varian 520/i computer. The DMA address bus and data bus are both brought to the connector (J5) located at the fourth card-slot.

The external device must provide a gate for each address bit and each data bit. The gates for each of the two busses must be capable of being collector OR'ed.

DMA Operating Modes

There are two modes of operation of the DMA port:

1. The mode used when the maximum access rate for any external equipment does not exceed 93kHz.
2. The mode used when the maximum access rate of any external equipment is greater than 93kHz.

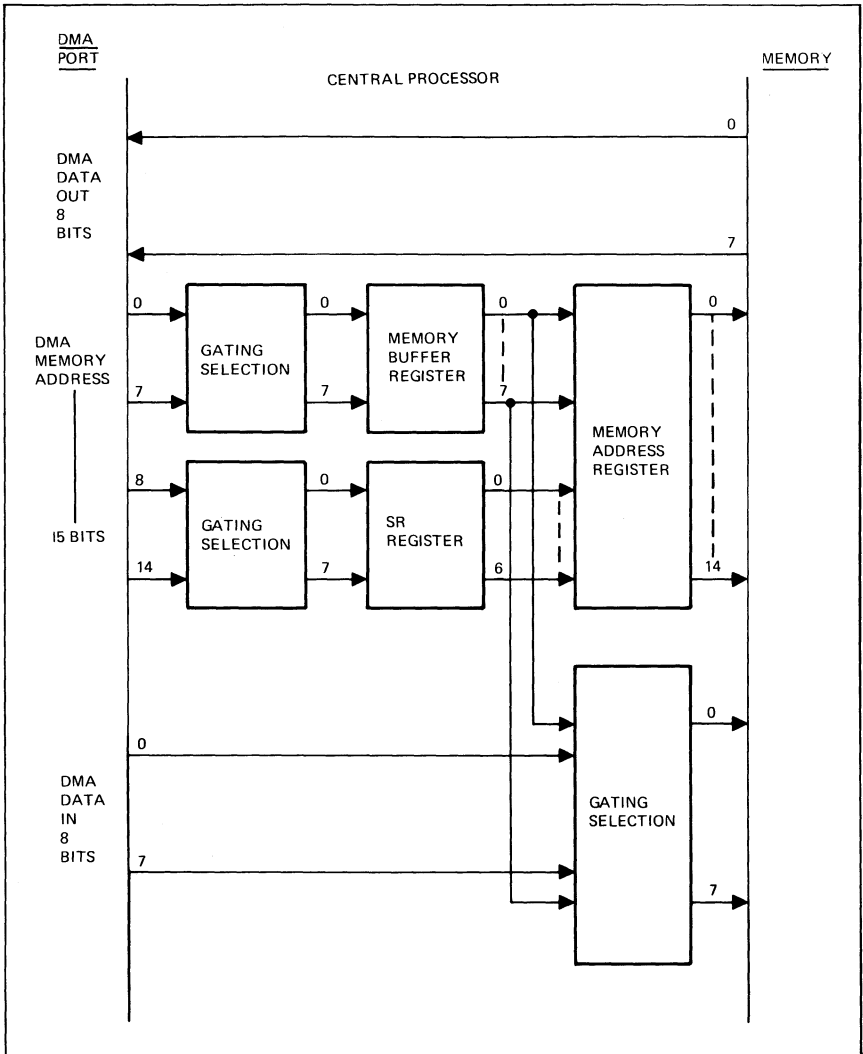


Figure 10-1. DMA Internal Logic

direct memory access

Less Than or Equal to 93kHz. A DMA request is not acknowledged until the instruction in process is complete. Since the maximum time required to perform any instruction, other than one which is indirect-addressed, is 9 microseconds (a branch and mark or a memory reference in precision 4), the maximum DMA rate is 93kHz.

This rate is determined by adding 9 microseconds for latency, 1.5 microseconds for DMA and 250 nanoseconds for the computer to setup the DMA memory address.

Any request rate up to and including this rate is acknowledged by the computer and access performed without any loss of data. A request rate greater than 93kHz is susceptible to errors depending on the instructions being executed by the program in the computer. If the request is greater than 93kHz, more than one DMA request could occur during the execution of certain instructions.

The maximum access rate is limited by the instruction in the program requiring the longest time to execute. A request rate greater than 93kHz could be achieved by careful programming to insure that DMA request occurs only during program instructions requiring less time than 9 μ sec to execute.

On the other hand, a branch-indirect-one-level instruction requires 6 microseconds to execute. For each additional level 2.75 microseconds are required. Since the DMA request is not acknowledged until the completion of the current instruction in process it is possible to have a latency time greater than 9 microseconds and consequently a maximum rate lower than 93kHz.

Greater Than 93kHz. For any device requiring a DMA rate greater than 93kHz, an I/O command initiated by the computer must be sent to the external equipment before DMA access is started.

This command holds the DMA request signal in the active state during the entire data transfer. At the completion of data transfer, the DMA request line is then made inactive.

By this means the data transfer rate of the DMA can be increased to 666,000 bytes per second. The only latency time is the time required for the DMA request to become active and for the computer to respond with a DMA acknowledge at the completion of the instruction in process. So long as the DMA request line is held active, the computer stays locked-up in the DMA data transfer mode. Every memory cycle is allocated to the DMA, thus prohibiting the computer's stored program from being executed. At the completion of the DMA data transfer, the interrupted program continues normally.

DMA Operation With Interrupts

Since neither a DMA request nor an interrupt request is acknowledged until the completion of the instruction in process, it can occur that the DMA and interrupt require a response from the computer at the completion of the same instruction. If this occurs, the following rules hold true:

DMA Request and Power Fail Interrupt. If at the completion of the instruction in process, the DMA request and power fail interrupt request signals are active, the

computer responds to the power fail interrupt and locks out any DMA transfer by holding the DMA acknowledge signal inactive. If while in a DMA lock-up mode, the power fail signal becomes active, the DMA data transfer is interrupted at the completion of the current access in process by causing the DMA acknowledge signal to become inactive.

DMA Request and Interrupts Other Than Power Fail. The DMA request has priority over all other interrupts. If any interrupt occurs during a DMA lock-up mode, it is not acknowledged until the completion of the DMA data transfer.

If the Varian 520/i includes the parity option, parity error detection is inhibited during a DMA. Parity is formed when inputting to memory but is not checked when reading out of memory.

User-Designed DMA Board

All the control logic to perform DMA functions in the computer, such as interrupting an operating program, cycle steal, arrested state and enabling address and data, are contained in the basic computer.

If a user decides to design his own DMA logic and install it in the computer, he has two choices open to him:

1. Build his logic on his own board.
2. Purchase a general-purpose socket board from Varian and build his logic on this board.

Figures 10.2 and 10.3 give the dimensions of the required board. The board material

should be Epoxy glass laminate with 2 oz. copper plating on both sides, FL-GF-062-C-2-2/2 per Mil-P-13949. The contacts should be hard gold plate, 0.00008 to 0.00010 inches thick.

The Varian general-purpose socket board comes in four versions:

Socket board — 1	87 14-pin sockets and 13 16-pin sockets
Socket board — 2	174 14-pin sockets and 26 16-pin sockets
Socket board — 3	261 14-pin sockets and 39 16-pin sockets
Socket board — 4	348 14-pin sockets and 52 16-pin sockets

The socket board is etched to allow the installation of 348 14-pin dual in-line IC sockets and 52 16-pin dual in-line sockets. The customer can locate the sockets on the board to fill his individual requirements. The sockets are interconnected using wire-wrap techniques. Connectors are located on the rear of the board to provide adequate cable interface.

Power Requirements

The dc power supply of the Varian 520/i is capable of supplying:

- 9 amperes of +5 V
- 1.6 amperes of +12 V
- 1.5 amperes of -12 V
- 2.1 amperes over range of +16 to +23 V

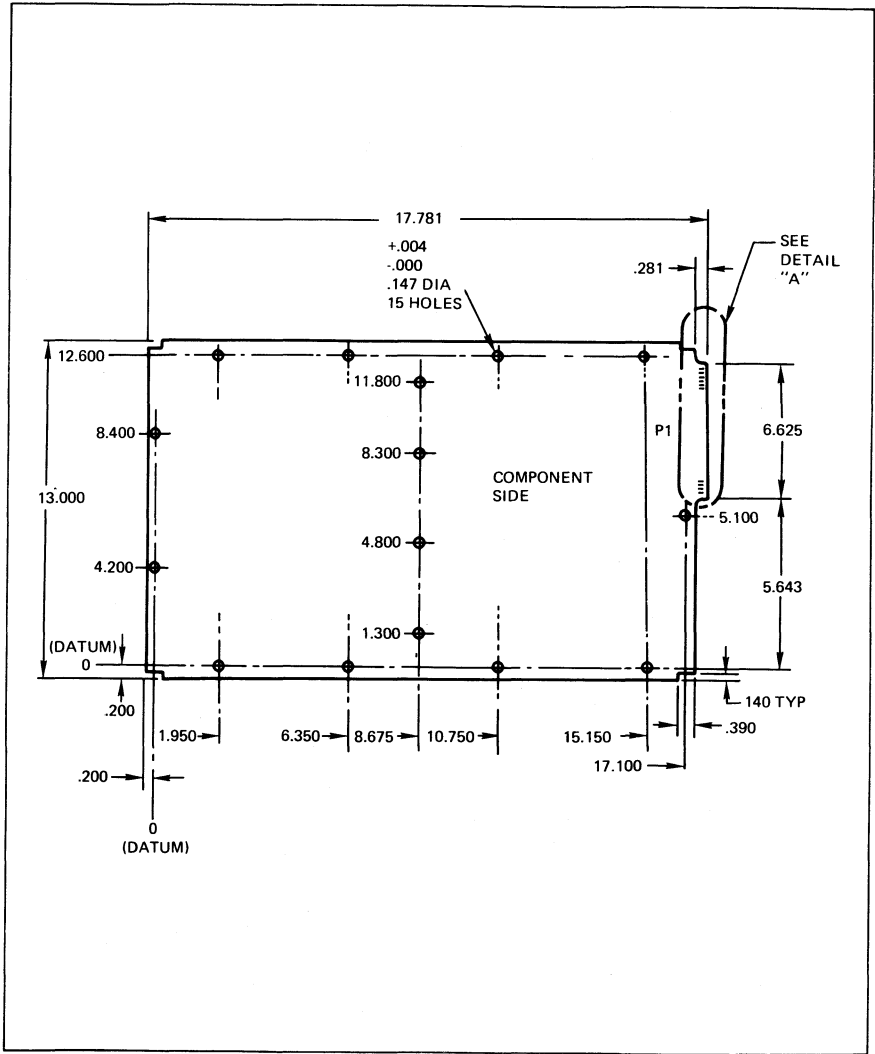


Figure 10-2.

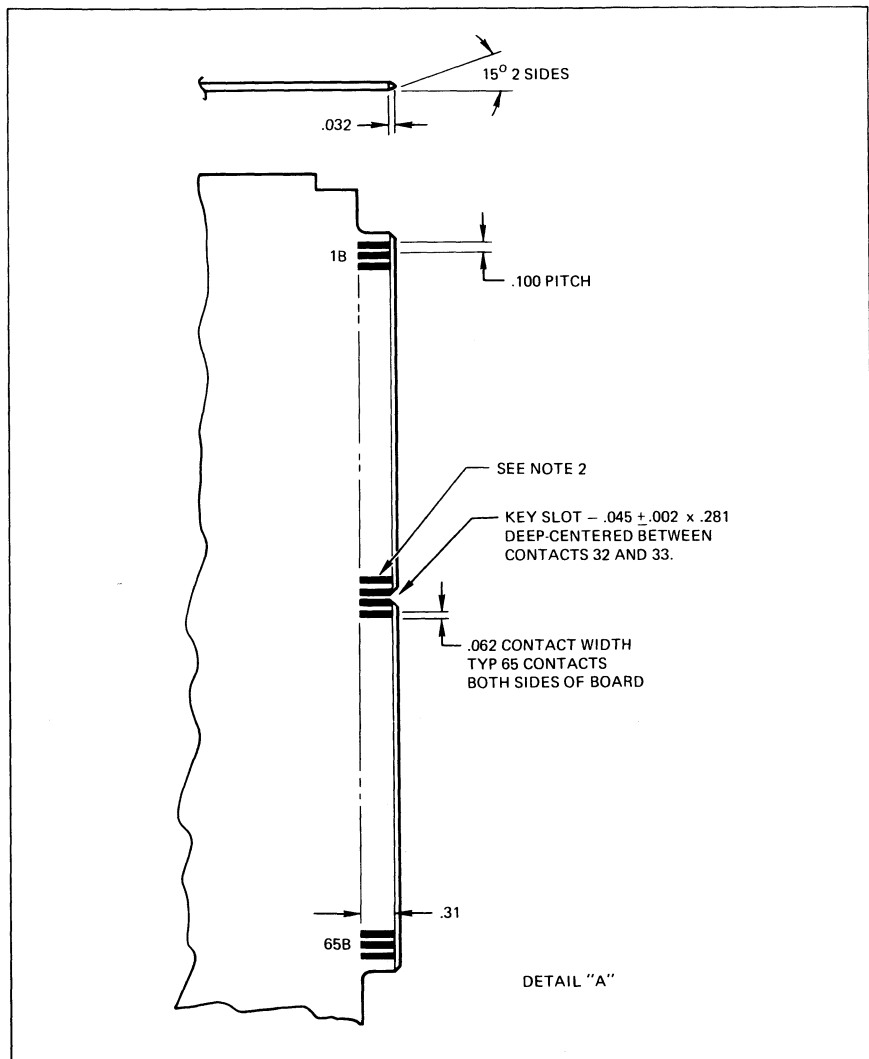


Figure 10-3.

direct memory access

The basic computer, Model 520/i-00, with teletype controller, requires the power listed below.

CPU	+5 V 5.420	+12 V 0.006	-12 V 0.145
Memory, 4 K	0.400 5.820 A	0.550 0.616 A	0.120 0.265 A

This leaves 3.180 amperes of +5 V
0.984 amperes of +12 V
1.235 amperes of -12 V
for all options contained in the computer.

The power requirements for Varian 520/i options are as follows:

a) 520/i-02 4K memory increment

+5 V	+12 V	-12 V
0.400 A	0.340 A	0.120 A

b) 520/i-03 memory parity option

0.232 A	N.A.	N.A.
---------	------	------

c) 520/i-14 Power failure/Restart

0.225 A	0.062 A	0.016 A
---------	---------	---------

d) 520/i-18 620/i peripheral adapter

1.416 A	0.100 A	N.A.
---------	---------	------

e) Paper Tape System

0.276 A	N.A.	N.A.
---------	------	------

f) DMA Port Line driver/receiver

0.158 A	0.075 A	0.097 A
---------	---------	---------

The systems configuration of the user therefore determines the amount of power available to him to build a DMA requesting device.

A DMA requesting device cannot be contained in the computer main frame if the computer also contains items d, e, and f, since these options are mounted in the fourth card slot position.

Interface Data, Direct to Internal Bus Structure

Listed in Figure 10-4 are the signals available to the external device at the connector in the main frame of the computer. The signals required for DMA operations are indicated by an X beside the signal.

The function of each of these signals is tabulated in Figure 10-5. Also shown is the logic level required to be active and either the number of loads on a signal or the fan-out capability of a signal. Fan out and load data refer to Series 74 TTL.

Since the external equipment connects directly to the internal bus structures of the Varian 520/i computer, the connection must follow the rules given below in order not to impair the operation of the computer.

Figure 10-6 shows the gating connection required to transfer a 15-bit address from the external equipment to the computer. The gates connecting to the IMB XX- and SRSTXX-bus must be open collector devices similar in characteristics to a Texas Instruments SN 7401. The external equipment is not required to provide a collector resistor on the busses.

Name	Pin #	Name	Pin #	Name	Pin #
Ground	1A	SRST00-	42A X	MB07A+	25B X
Ground	2A	SRST01-	43A X	MB06A+	26B X
Ground	3A	SRST02-	44A X	MB05A+	27B X
-12VDC	4A	SRST03-	45A X	MB04A+	28B X
+12VDC	5A	SRST04-	46A X	MB03A+	29B X
+1000A+	8A	SRST05-	47A X	MB02A+	30B X
T125-	9A	SRST06-	48A X	MB01A+	31B X
INIT-	10A	SRST07-	49A X	MB00A+	32B X
SERP-	11A	BTO+	50A	FULDSB-	43B
SRLDFU-	15A	SEN+	51A	FLDLSB-	44B
SRLDFL-	16A	FUN+	52A	MDAT00+	46B X
PA100-	17A	BTI+	53A	MDAT01+	47B X
PA101-	18A	SBUS00-	54A	MDAT02+	48B X
PA102-	19A	SBUS01-	55A	MDAT03+	49B X
PA103-	20A	SBUS02-	56A	MDAT04+	50B X
PA104-	21A	SBUS03-	57A	MDAT05+	51B X
PA105-	22A	SBUS04-	58A	MDAT06+	52B X
PA106-	23A	SBUS05-	59A	MDAT07+	53B X
PA107-	24A	SBUS06-	60A	SR00+	56B
PA108-	25A	SBUS07-	61A	SR01+	57B
PA109-	26A	SR07+	62A	SR02+	58B
PA110-	27A	Ground	63A	SR03+	59B
IOP+	30A	Ground	64A	SR04+	60B
MBLDDM+	31A X	Ground	65A	SR05+	61B
DMWRIT+	32A X	+5VDC	1B	SR06+	62B
DMARQ-	33A X	+5VDC	2B	+5VDC	63B
IMB00-	34A X	+5VDC	3B	+5VDC	64B
IMB01-	35A X	DMACK+	6B X	+5VDC	65B
IMB02-	36A X	RWCK+	7B X		
IMB03-	37A X	DMA0S+	8B		
IMB04-	38A X	MMLDDM+	9B X		
IMB05-	39A X	IOLDDM+	10B X		
IMB06-	40A X	IO-	11B		
IMB07-	41A X				

Figure 10-4. Pin Assignments, Connector J5

direct memory access

Name	Mnemonic	J5 Pin	Active Level	Number of Loads	Function
DMA request	DMARQ-	33A	0 vdc	2	Indicates to the 520/i that an external device requires a direct memory access.
DMA write	DMWRIT+	32A	+4 vdc	1	Indicates that a write operation is requested. The complement of this signal indicates a read operation.
DMA acknowledge	DMAACK+	6B	+4 vdc	1	Indicates that the DMA request has been accepted.
Input to MB register	IMB00- IMB01- IMB02- IMB03- IMB04- IMB05- IMB06- IMB07-	34A 35A 36A 37A 38A 39A 40A 41A	0 vdc	1	Bits 00-07 of the DMA address buffered and enabled by MBLDDM. Sets the MB register, which temporarily stores DMA address 00-07.
Input to SR register	SRST00- SRST01- SRST02- SRST03- SRST04- SRST05- SRST06- SRST07-	42A 43A 44A 45A 46A 47A 48A 49A	0 vdc	1	Bits 08-14 of the DMA address buffered and enabled by MBLDDM. Sets the SR register, which temporarily stores DMA address 08-14.
DMA data to the memory	MB07 A+ MB06A+ MB05A+ MB04A+ MB03A+ MB02A+ MB01A+ MB00A+	25B 26B 27B 28B 29B 30B 31B 32B	0 vdc	1	DMA data word buffered and enabled by MMLDDM. Applied directly to the memory.

Figure 10-5. DMA Signals To and From the Main Frame (Sheet 1)

Name	Mnemonic	J5 Pin	Active Level	Number of Loads	Function
Data from the memory	MDAT00+ MDAT01+ MDAT02+ MDAT03+ MDAT04+ MDAT05+ MDAT06+ MDAT07+	46B 47B 48B 49B 50B 51B 52B 53B	+4 vdc	1	Eight-bit data word from the memory.
Load DMA address in	MBLDDM+	31A	+4 vdc	10	Enables the DMA address to be loaded into the MB and SR registers.

Figure 10-5. DMA Signals To and From the Main Frame (Sheet 2)

The DMA address (DMADXX) presented by the external device is at the high level, +4 V, when it is in the true condition.

The control signal, MBLDDM, provided by the computer should be amplified by the external equipment so as to distribute the loads on this signal.

The data for DMA transfer may originate either at the computer or at the external equipment.

Figure 10-7 shows the data lines, called MDAT00-07, used to transfer data from the computer to the external equipment. Data from the computer memory is held static between each memory cycle and the lines are directly from the memory. Only one load should be applied to each output. The data lines are at the +4 V level when designating a true condition.

The control term, IOLDDM, is true only during a DMA transfer and during the time the data lines are static. The data lines are in a state of change during memory-access period.

Figure 10-7 also shows the data lines, called DMB00-07, used to transfer data to the computer from the external equipment. The data lines are at the 1 V level when designating a true condition. The control term MMLDDM+ is true only during a DMA transfer and is at the +4 V level when true or active.

The logic gates used to OR the DMA data to the computer data bus to the memory must be capable of being collector OR'ed and are similar in characteristics to a Texas Instruments 7401 N. (Open collector devices.) The external equipment does not provide an external collector resistor on the bus.

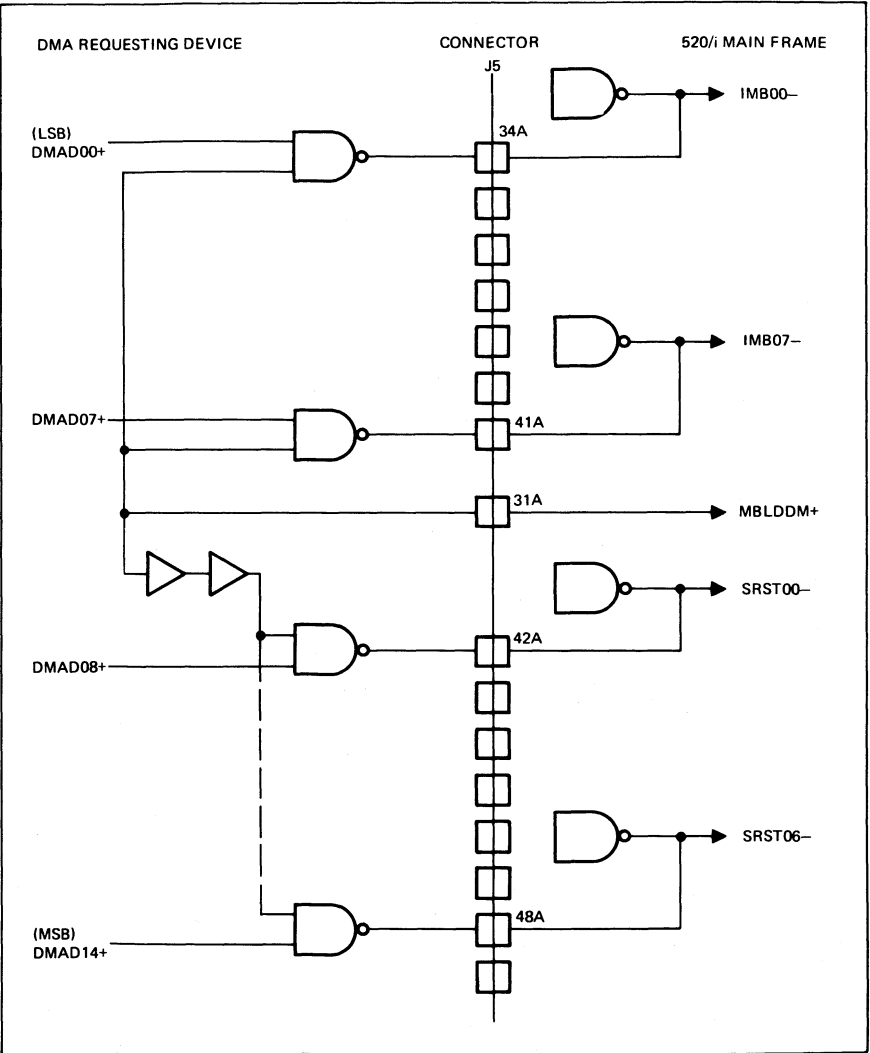


Figure 10-6. DMA Address Connections to the Varian 520/i Mainframe

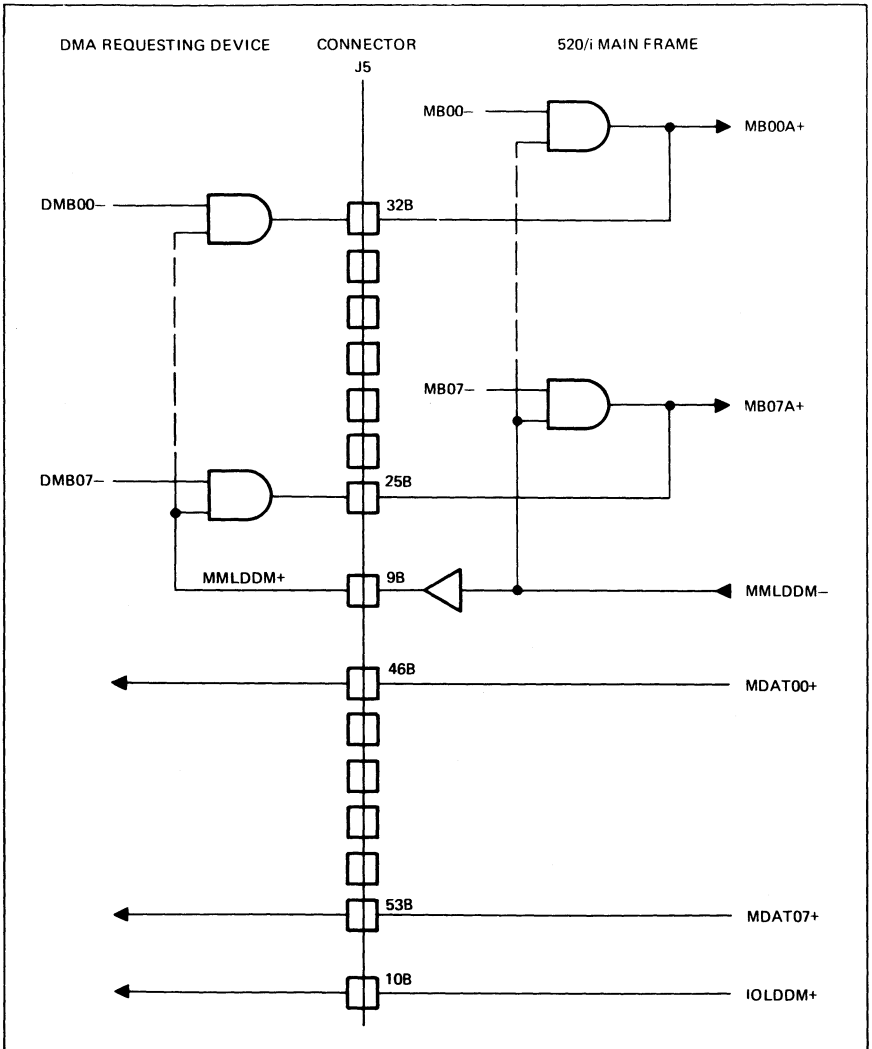


Figure 10-7. DMA Data To and From the Varian 520/i Mainframe

Interface Data, Varian-Supplied Drivers and Receivers

The Varian 520/i-12 option consists of an 18.375-by-4.5-inch circuit board on which are mounted drivers and receivers to effect DMA data transfers. Plugs connecting to the board are used to interconnect the external devices in accordance with Figures 10-8, 10-9, and 10-10.

Timing Data

Figure 10-11 shows the timing of the DMA signals, both on the internal bus and at the DMA Port.

The times shown for the DMA Port include a 40 nanosecond delay in the cable and time losses in the drivers (or receivers) in the users' device. It is assumed that the Varian 520/i-12 driver/receiver circuits are used. Time reference in the DMA requesting device is to the receiver outputs or to the input to the SN7401N preceding the driver.

The time designations at the top of Figure 10-11 are based on the computer timing, which is derived from an 8MHz clock. A complete memory cycle is usually 1.5 microseconds. In the cycle preceding a DMA cycle, or a group of DMA cycles, a 250

nanosecond pause is inserted. (This pause also occurs during an interrupt.)

Read/Write Clock provides a time reference to the Varian 520/i. Normally it is a symmetrical square wave with a 1.5 microsecond period. A pause of 250 nanoseconds is added to the second phase on the first cycle only.

DMA Request is required for the period indicated by the low level of the waveform. The shaded area shows how it may be extended at the user's option. If it is held too long, a second DMA cycle may be started.

DMA Acknowledge gives the user the first indication that he has the next memory cycle. It remains as long as DMA cycles persist and is removed after the request is removed.

DMA Address waveform shows the minimum time required for stable data on these lines. It can be seen that gating address data with DMA Acknowledge could result in incorrect address data during the sampling period. It is preferable to apply address data with the DMA Request.

Name	Plug-Pin	Name	Plug-Pin
	P2-01	DMAAD01	P2-17
GND	2	GND	18
	3	DMAAD02-	19
GND	4	GND	20
	5	DMAAD03-	21
GND	6	GND	22
	7	Key Slot	23
GND	8	GND	24
	9	DMAAD04-	25
GND	10	GND	26
	11	DMAAD05-	27
GND	12	GND	28
DMAWRT-	13	DMB00-	29
DND	14	GND	30
DMAAD00-	15	DMAAD14-	31
GND	16	GND	32
DMAAD13-	33	DMAAD10-	39
GND	34	GND	40
DMAAD12-	35	DMAAD09-	41
GND	36	GND	42
DMAAD11-	37	DMAAD08-	43
GND	38	GND	44
GND	P5-01	DMDAT01-	P5-22
DMAAK-	2	GND	23
GND	3	DMDAT00-	24
DMARWC-	4	GND	25
GND	5	DMB07-	26
DMASPI-	6	GND	27
GND	7	DMARQ-	28
DMDAT07-	8	DMAAD06-	29
GND	9	GND	30
DMDAT06-	10	DMB05-	31
GND	11	GND	32
DMB06-	12	DMB04-	33
GND	13	GND	34
DMDAT05-	14	DMB03-	35
GND	15	GND	36
DMDAT04-	16	DMB02-	37
GND	17	GND	38
DMDAT03-	18	DMB01-	39
GND	19	GND	40
DMDAT02-	20	DMAAD07-	41
GND	21	GND	42
			43
		GND	44

Figure 10-8. Plug-Pin Assignments, Varian 520/i-12 DMA Option

direct memory access

Name	Mnemonic	Function
DMA Request	DMARQ-	Indicates to the computer that an external device requires a direct memory access.
DMA Write	DMAWRT-	Indicates that a Write operation is requested. The complement of this signal indicates a Read operation.
DMA Address In	DMAAD 00-14-	15-bit memory address from the external device
DMA Data In	DMB 00-07-	Eight-bit data word from the external device
DMA Data Out	DMDAT 00-07-	MDAT 00-07 buffered and enabled to the external device (DMA bus)
DMA Acknowledge	DMAAK-	DMAACK+ buffered to the external device.
Read/Write Clock	DMARWC-	RW Clock buffered to the external device.
Spare	DMASPI-	DMAOS buffered to the external device

Figure 10-9. Signal Functioning, Varian 520/i-12 DMA Option

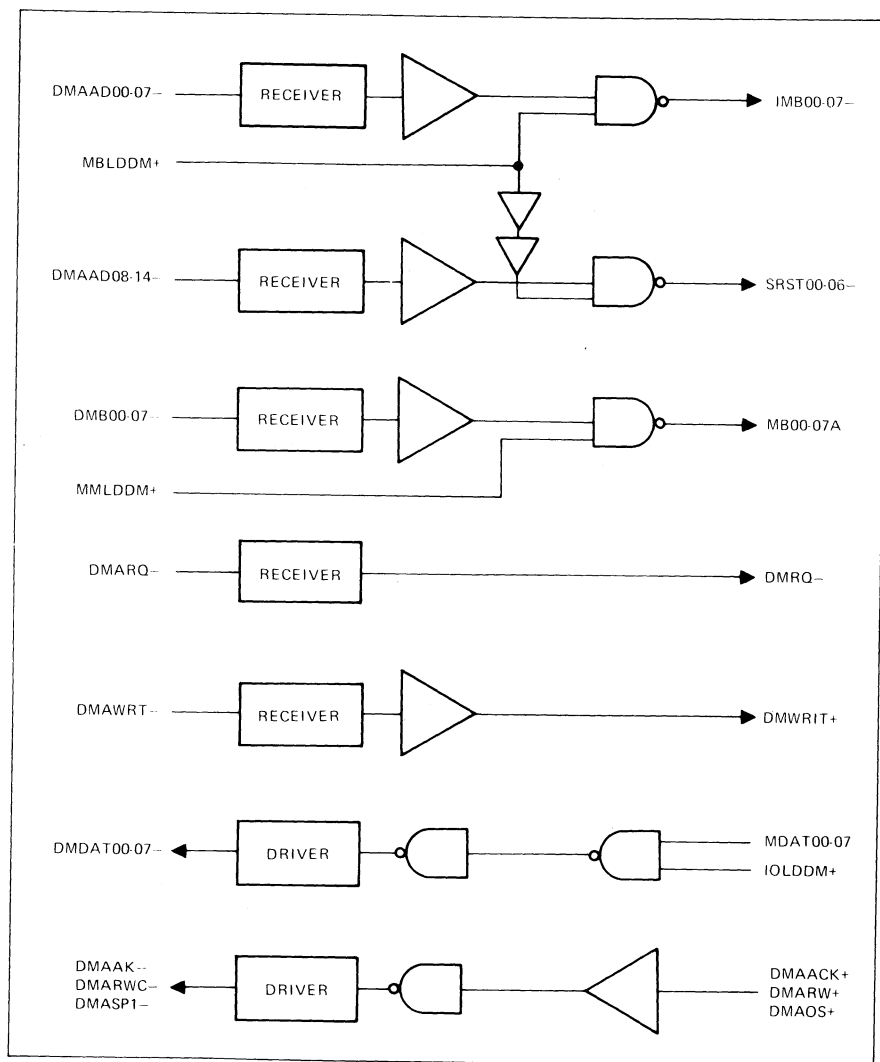


Figure 10-10. DMA Port Line Driver/Receiver Block Diagram

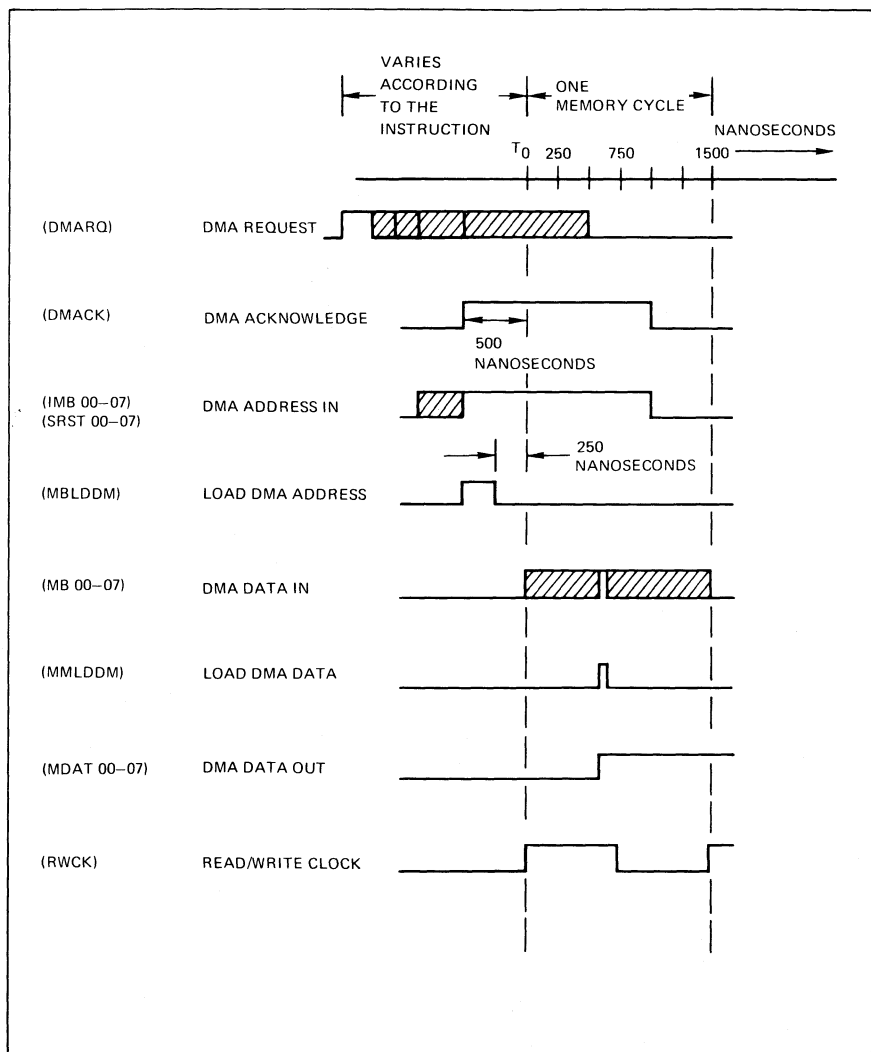


Figure 10-11. DMA Port Timing

SECTION 11 – OPERATOR'S CONSOLE

The "operator's console" of the Varian 520/i consists of the front panel of the main-frame chassis and the teletype that is normally included, as a minimum recommended configuration. These two elements provide all the controls and displays needed to program, operate, and maintain the computer.

The controls and indicators on the front panel of the Varian 520/i are shown in Figure 5-1 and a description of their functions is given in the following paragraphs.

Data Display Indicators. A set of 16 binary indicators constitute the principal data display. The source of the display is switch selectable and can include a memory location, an accumulator, an index register, or a program counter.

Data Entry Switches. A set of 16 binary switches is used to load the display register with a data word (or instruction) which is then transferred to memory or to one of the operational registers.

CLEAR DISPLAY Switch. The CLEAR DISPLAY switch clears the display register and the display to all zeros.

Overflow Indicators 01 and 02. Two indicators are used to reflect the status of the arithmetic overflow. For convenience of display of individual memory locations the use of the STEP switch in conjunction with

the OR switch is outlined in Figure 14-8. Under these conditions, the other uses for the STEP switch are inoperative flip-flops in each environment. Indicator 01 denotes an overflow in the interruptable environment; Indicator 02, in the non-interruptable environment.

RUN Indicator and Switch. The RUN indicator illuminates during the time the machine is actively executing an instruction. When the RUN switch is depressed, the machine starts executing instructions, starting with the contents of the current-program-counter location.

STEP Indicator and Switch. The STEP indicator illuminates when the machine is not running, but the power is on, and the front-panel controls and indicators are active and may be used for register display and entry. The STEP switch is used to single-step instructions for program or hardware check. It is also used to halt a program in progress; depression of the STEP switch causes the machine to leave the run condition and enter the step condition.

RESET Switch. Momentary activation of the RESET switch returns all control flip-flops in the machine to an initialized condition. The environmental status is returned to the Interruptable Environment. The operand precision is set to one byte (8 bits). The teletype and other peripheral devices are initialized.

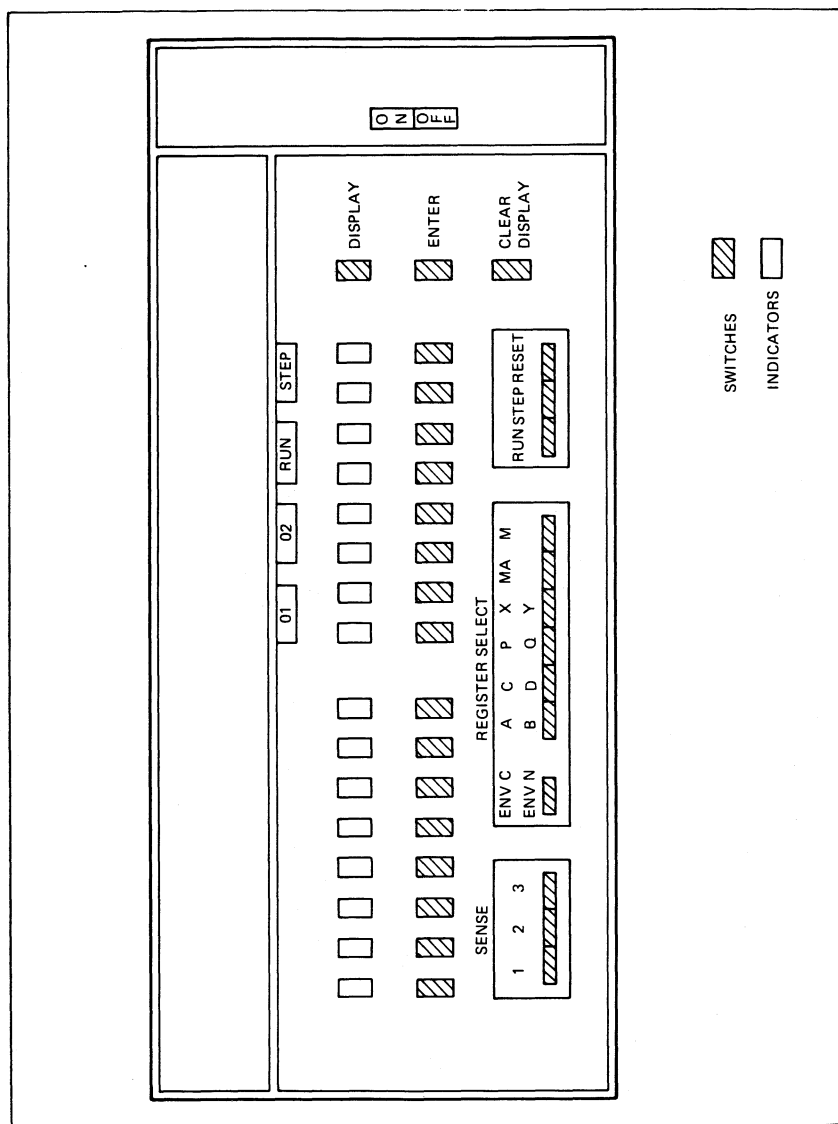


Figure 11-1. Varian 520/i Front Panel

REGISTER SELECT Switches. The seven REGISTER SELECT switches determine the register that is to be displayed, or the register that is to be loaded with the data represented by the Data Entry Switches.

The switches are:

ENV C/ENV N	Determines whether the selected accumulator, index register or program counter is for the current or non-current environment.
A/B	Selects upper accumulator (current or non-current),
C/D	Selects lower accumulator (current or non-current),
P/Q	Selects program counter (current or non-current),
X/Y	Selects index register (current or non-current).

MA	Selects memory address register (cannot be displayed).
----	--

M	Selects memory data register.
---	-------------------------------

DISPLAY Switch. The contents of a selected register or memory location are displayed by depressing the DISPLAY switch. The contents are altered by clearing the display and entering new data via the data entry switches and the ENTER switch.

ENTER Switch. The data entry switches set the display register; pressing the ENTER switch transfers the contents of the display register to the selected register.

Sense Switches 1, 2 and 3. The three Sense Switches allow program action. The condition of each switch can be interrogated by the program and decisions made in accordance with the Sense Switch settings.

ON/OFF Switch. The power ON/OFF switch controls the application of ac power to the power supplies in the Varian 520/i. Pressing the ON button resets the computer to an initialized condition.

SECTION 12 -- TELETYPE AND CONTROLLER

A modified ASR 33TC teletypewriter is the standard peripheral device for communication with and control of the Varian 520/i computer. Data exchange between the Varian 520/i and the teletype is via a teletype controller installed in the main frame of the computer.

The teletype and controller provide a ready means of transferring data in and out of the Varian 520/i. Data input is from the keyboard and the paper tape reader. Data output is to the printer and the paper punch. (Note: Because of the mechanical connection between the punch and the printer, it is impossible to punch paper tape without also getting a response from the printer.)

The controller enables the teletype to send data from the teletype keyboard in ASCII code or to send data from the teletype paper tape reader (in binary code, 8-bits per word) to the teletype buffer register. The controller also enables the teletype to receive data from the teletype buffer register for printing (ASCII data is needed to print information) or for punching and printing (the punch records the bits in the lower 8-bits of the accumulator; the printing will not, however, necessarily be understandable). The hexadecimal representations of all acceptable ASCII characters are given in Figure 12-1.

Figure 12-2 shows a simplified block diagram of the Varian 520/i and an ASR 33TC teletype. Data transfer between the lower 8-bits of the accumulator and the teletype buffer register is under the basic control of the CPU, as described in Section 3 (also see BTI and BTO instructions, as described in Appendix). This section deals principally, therefore, with how the controller transfers information between the teletype and the teletype buffer register.

Functional Description

A block diagram of the teletype controller and the teletype are shown in Figure 12-3.

The command decoding block decodes the instructions to the teletype controller and then causes the other blocks to perform their function.

The teletype interrupt blocks (read or write) send low, ground-level signals to the basic control logic when the teletype read or write interrupts are set (see page 26-20).

The input or output buffer-ready detector block sends high level signals to the CPU control logic when the teletype can receive (output buffer-ready flag) or send (input buffer-ready flag) data. These high level signals may be detected by sense instructions.

LOWER CASE CHARACTERS				SPECIAL CHARACTERS	
1	B1	(LINE FEED)	8A	(WRU)	85
2	B2	(RETURN)	8D	(TAPE)	92
3	B3	A	C1	(TAB)	89
4	B4	S	D3	(XOFF)	93
5	B5	D	C4	(EOT)	84
6	B6	F	C6	(RU)	86
7	B7	G	C7	(BELL)	87
8	B8	H	C8	(VT)	88
9	B9	J	CA	(FORM)	8C
0	B0	K	CB	---	91
:	Ba	L	CC	---	97
-	AD	:	BB	---	94
(ALT MODE)	FD	(RUB-OUT)	FF	---	99
Q	D1	Z	DA	---	95
W	D7	X	D8	---	8F
E	C5	C	C3	---	90
R	D2	V	D6	---	81
T	D4	B	C2	---	88
Y	D9	N	CE	---	9A
U	D5	M	CD	---	98
I	C9	'	AC	---	83
ø	CF	.	AE	---	96
P	D0	/	AF	---	82
		(SPACE)	A0	---	8E

UPPER CASE CHARACTERS	
!	A1
"	A2
#	A3
\$	A4
%	A5
&	A6
'	A7
(	A8
)	A9
*	AA
=	BD
←	DF
[	DB
\	DC
@	C0
+	AB
†	DE
]	DD
<	BC
>	BE
?	BF

Figure 12-1. Hexadecimal Representation of all Acceptable ASCII Characters

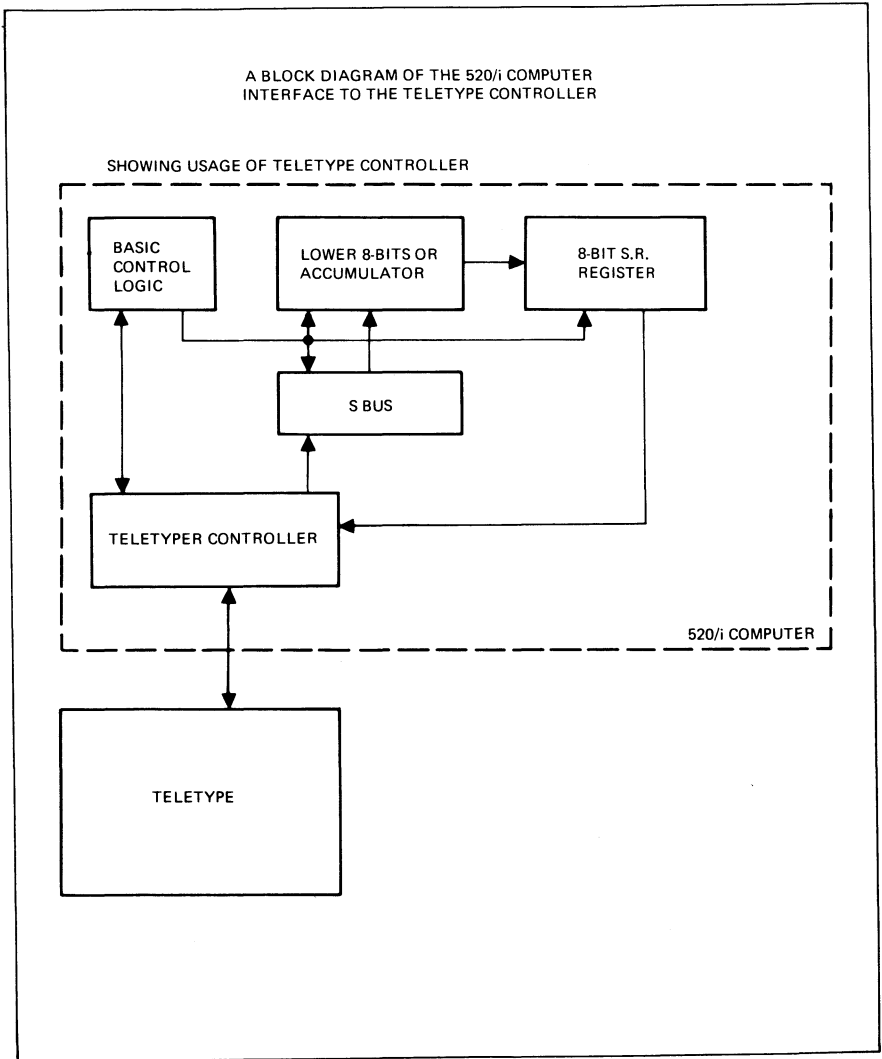


Figure 12-2. Block Diagram of the Computer Interface to the Teletype Controller

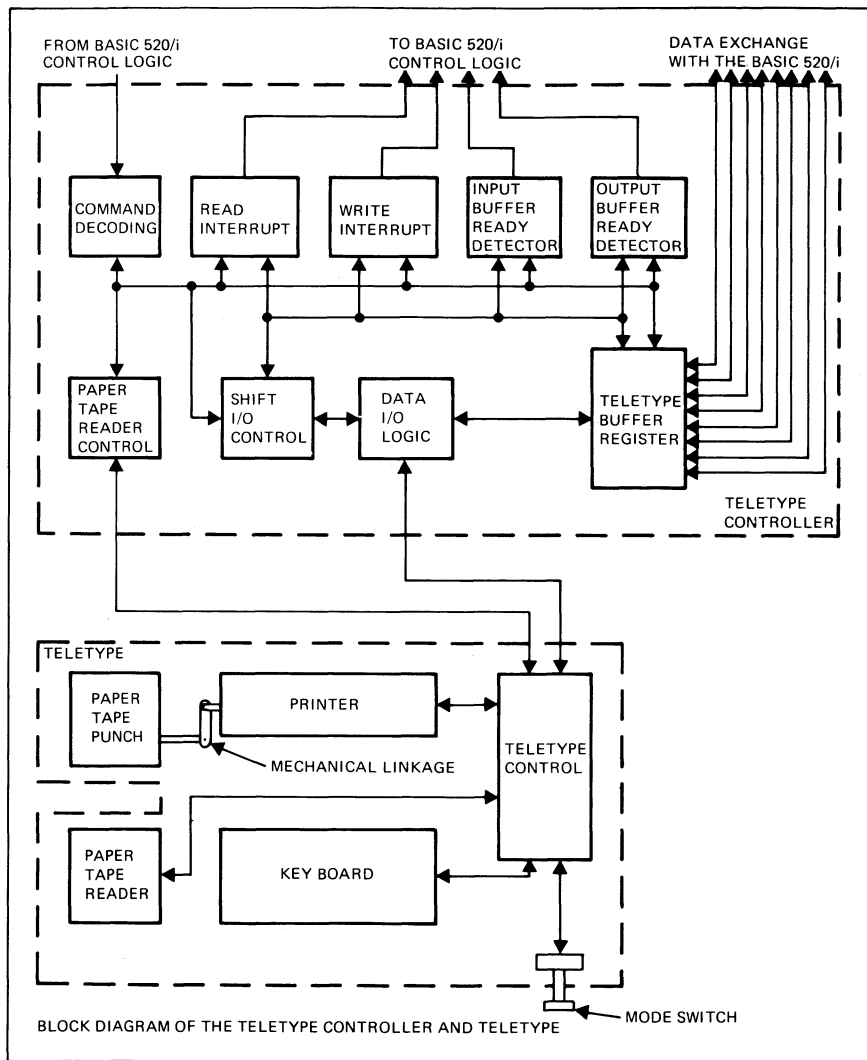


Figure 12-3. Block Diagram of the Teletype Controller and Teletype

The **shift I/O control** block controls the shifting of data in and out through the teletype buffer register.

The **paper tape reader** control block turns the paper tape reader on and off.

The **data I/O logic** block uses inputs from the shift I/O control and from the teletype to control the direction of data flow, out to the teletype or in from the teletype.

The **teletype buffer** register stores data as it is shifted in from or out to the teletype. Data transfer between the teletype and the 520/i is in bit serial form. Data transfer from the teletype buffer to the rest of the 520/i is parallel (8-bits).

Teletype Instructions

The following paragraphs detail the instructions that affect the teletype controller. The instructions are listed in Figure 12-4.

FUN 21 Reset TTY Controller

This instruction sets the output buffer ready flag, resets the input buffer ready flag, halts the paper tape reader, and disables both the read and write enable interrupt conditions.

Caution must be exercised when executing this instruction when the teletype controller is presently processing previous data, as this activity will be terminated prior to completion. (To prevent unintentional termination of data processing a SEN 21 instruction should be used.)

FUN 61 Enable Read Interrupt

This instruction enables the read interrupt mask. Thus when the input buffer ready flag is set (by a completion of a data transfer to the Varian 520/i from the teletype) a teletype interrupt indication will be generated.

This command is used to allow the computer to perform other operations while the teletype controller is handling data transfers from the teletype. The computer is not tied up, therefore, in a sense loop waiting for the data to be sent by the slow teletype.

FUN 01 Enable Write Interrupt

This instruction enables the write-interrupt mask. Thus when the write-buffer-ready flag is set, a teletype interrupt indication will be generated.

FUN C1 Read One Character

This instruction causes the paper-tape reader to advance one character and 8-bits of data to be transferred to the teletype controller buffer. The data may then be input to the C register by executing a BTI 01 instruction. The data received is that positioned directly over the read head prior to the execution of the read-one-character instruction. (Note: The BTI 01 Instruction should be preceded by a sense-input-buffer-ready instruction.)

FUN 41 Read Continuous

This instruction causes the paper tape reader to advance frames in a continuous

Symbolic	Hexadecimal	Action
Function Out		
FUN 21	3321	Reset TTY Controller
FUN 61	3361	Enable Read Interrupt
FUN 01	3301	Enable Write Interrupt
FUN C1	33C1	Read One Character
FUN 41	334 1	Read Continuous
Data Transfer		
BTI 01	3101	Transfer Data From TTY Buffer to C Register
BTO 01	3001	Transfer Data From C Register to TTY Buffer
Sense		
SEN 21	3221	Sense Output Buffer Ready
SEN 41	3241	Sense Input Buffer Ready

Figure 12-4. Input/Output Instructions

manner until terminated by a reset teletype controller instruction or by depressing SYSTEM RESET on the computer control panel. Whenever a frame of data is shifted into the teletype buffer, the input-buffer-ready flag is set and may be detected by a SEN 41. The data may then be transferred to the C register by executing a BTI 01 instruction.

When paper tape motion is stopped by executing the reset teletype controller instruction, the paper tape advances one frame beyond the last processed frame

and the input-buffer-ready flag is set. If a sense-input-buffer-ready instruction, followed by a BTI 01 instruction, is executed after issuing a reset teletype controller instruction, the additional frame of data is received and the teletype is brought to a quiescent state. (Note: if no action is taken following the reset teletype controller instruction the last frame of data is held in the teletype buffer and the input-buffer-ready flag is set. Thus a subsequent request for teletype input (from the keyboard, for example) is satisfied with the data held in the teletype buffer, making it possible

to get erroneous data. To keep this from occurring, the FUN 41 instruction should be followed by a sense-input-buffer-ready instruction and a BTI 01. Otherwise the first data input from the teletype following a FUN 42 should be considered erroneous.)

SEN 21 Sense Output Buffer Ready

This is a sense and skip instruction. The execution of the instruction causes the two bytes of the program following 21 to be skipped whenever the teletype controller buffer is ready to accept output data. This condition may be caused by executing a reset, teletype controller instruction, or the absence of any outstanding input or output activity.

SEN 41 Sense Input Buffer Ready

This is a sense and skip instruction. The execution of the instruction causes the two bytes of the program following the 41 to be skipped whenever the teletype controller buffer has an input character prepared. This condition will be caused when the data transfer to the teletype buffer is completed after having previously issued a Read-One-Character instruction, or a Read-Continuous instruction, or depression of a key on the teletype keyboard.

Programming Examples

Teletype Character Input Subroutine: Upon hitting a key, the following subroutine inputs an ASCII character into the lower 8-bits of the C register. Note that depressing

a key on the teletype keyboard does not automatically cause the character to print on the teletypewriter. The data rate may be 0 to 10 characters per second.

0B27	32	SEN	Check Input Buffer
0B28	41	41	Ready
0B29	F3	BRU 0B	BRU Back to 0B27
0B2A	27	27	If buffer not ready
0B28	31	BTI	Byte transfer In
0B2C	01	01	From Teletype
0B2D	??	BRU	BRU to your
0B2E	??		Program

Output One Character Subroutine: The following subroutine causes the character contained in the least significant 8-bits of the C register to be output to the teletype printer. If the punch is mechanically turned on, the corresponding character is also punched on paper tape. The data rate may be from 0 to 10 characters per second.

0B27	32	SEN	Sense Output Buffer
0B28	21	21	Ready
0B29	F3	BRU 0B	BRU Back to 0B27
0B2A	27	27	If buffer not ready
0B2B	30	BTO	Byte Transfer Out
0B2D	01	01	to teletype
0B2E	??	BRU	BRU to your
0B2F	??		program

Echo Teletype Character Subroutine: The following subroutine combines the input-one-character subroutine and the output-one-character subroutine. When a teletype key is depressed, the character is printed (200 milliseconds later). Because of the interlacing of the input and output subroutines, this routine can only handle 0 to 5 characters per second. Typing in excess of

teletype and controller

this rate may cause erroneous data to be received.

0B27	32	SEN	Check Input Buffer
0B28	21	41	Ready
0B29	F3	BRU 0B	BRU Back to 0B27
0B2A	27	27	If Buffer not ready
0B2B	31	BTI	Byte Transfer In
0B2C	01	01	From Teletype
0B2D	32	SEN	Check Output Buffer
0B2E	21	21	Ready
0B2F	F3	BRU 0B	BRU Back to 0B2D
0B30	2D	2D	If Buffer not Ready
0B31	30	BTO	Byte Transfer Out
0B32	01	01	to teletype
0B33	F3	BRU 0B	BRU Back to
0B34	27	27	Beginning

Read One Character Subroutine: The following subroutine advances the paper tape reader one character and transfers the received character into the least significant 8-bits of the C register. The output-buffer-ready sense is first checked in order not to interfere with any previous output activity that may not yet be completed.

0000	32	SEN	Check output buffer
0001	21	21	ready

0002	E0	BRU 00	BRU Back to 0000
0003	00	00	If buffer not ready
0004	33	FUN	Read One Byte
0005	C1	C1	From paper tape
0006	32	SEN	Sense Input
0007	41	41	Buffer Ready
0008	E0	BRU 00	BRU Back to 0006
0009	06	06	If Buffer Not Ready
000A	31	BTI	Byte Transfer In
000B	01	01	from Teletype
000C	??	BRU	Go to your
000D	??	??	program

Physical Description

The teletype controller is located within the main frame of the computer on the CPU module. It is a part of the DM 141 and DM 155 boards and has the same voltage requirements and environmental requirements as the Varian 520/i.

The computer is supplied with a keyed cable connector (J1 of the DM 155 board) for interconnection with the ASR 33TC. The teletype comes with the cable attached to it. Therefore it is only necessary to plug the cable into the rear of the computer to complete the interconnection.

SECTION 13 — PAPER-TAPE PUNCH/READER

The paper-tape subsystem for the Varian 520/i is incorporated in a single chassis which contains a paper-tape punch, photo-electric reader, power supplies, power control, punch magnet drivers and all necessary logic.

The subsystem chassis is slide-mounted to provide accessibility to the punch tape supply and the punch paper collector, both of which are located behind the front panel. The panel is 10-1/2 inches high and conforms to standard 19-inch RETMA dimensions (see Figure 13-1).

Controls located on the front panel provide for feeding tape through the punch and for off-line operation of the tape reader.

Paper Tape Punch

The paper tape punch provides the ability to record computer output data of up to eight bits per character on a one inch wide paper tape with a data density of ten characters per inch.

The punch is an electromechanical device, operating from a continuously driven input shaft. It is a synchronous device having a magnetic pickup which produces synchronizing pulses to trigger the control circuits.

The punch perforates the tape by means of punch pins in a die block each time a code

combination is transmitted to the punch and a command to record the data is given. A feed hole is automatically perforated in the tape along with the data. The feed hole is used to transport the tape through the punch by means of a sprocket wheel and ratchet assembly.

The punch magnet drivers are standard Varian solenoid drivers, mounted four to a printed circuit card. Two printed circuit cards carry the data drivers. An identical ninth driver for the feedhole magnet is mounted on the control logic card, which also carries synchronizing logic, a timing-signal-shaper circuit, and the punch-up-to-speed detector logic.

The punch is mounted behind and to the right of the front panel of the chassis. Between the punch and the front panel is a fanfold tape bin which protrudes through the front panel for ease of removal of punched tape. This bin accepts up to 200 feet of punched paper tape from the punch.

The punch is rubber shock mounted to reduce vibration and noise. The punch drive motor is mounted to the rear and below the punch. Power is transmitted from the motor to the punch via a cog-type drive belt.

The supply of paper tape to the punch can be either fanfold or roll. Provisions are made to hold a full 1000 foot box of fold tape in a bracket. This same bracket has provisions

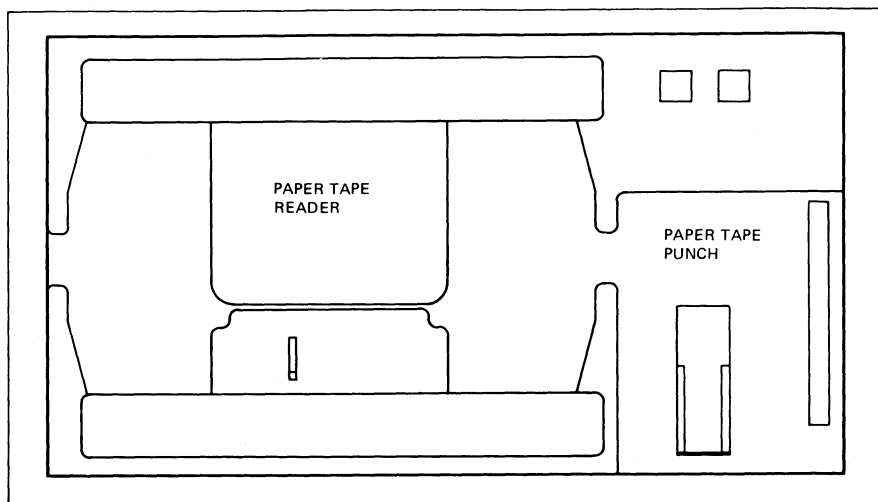


Figure 13-1.

for holding an 8-inch tape supply reel of 1000 feet of roll tape.

Provisions are made for mounting tape guides as an option for reel take-up of tape from the punch. The take-up reel must be on another panel as there is physically no room for such a device on the chassis panel.

A manual means of feeding tape through the punch is provided.

Punched Paper Tape Reader

The paper tape reader is a unidirectional tape transport unit which utilizes a continuously driven capstan to transport the tape.

Tape movement is controlled by a solenoid operated pinch roller. Operation of the solenoid closes the gap between the paper

tape and the capstan and thereby causes the tape to feed with the driven capstan.

Stopping the tape requires de-energizing the pinch roller solenoid and energizing a brake solenoid. Operation of the brake solenoid closes the gap between the brake shoe and a fixed surface, thereby stopping the tape which is between the two surfaces.

The tape is transported from left to right facing the front of the tape reader.

Nine photovoltaic cells are used to sense the perforations in the tape. A constant light source provides a focused beam which covers the area of the photocells. When a hole in any given track on the tape appears over the top of the photocell block, the light shining through the hole energizes the cell.

To obviate problems in reading the code holes caused by differences in punch alignment positions and skew of the tape, the data outputs are gated with the output of the sprocket channel. The sprocket hole being smaller in size than the data holes provides a window which comes true after and goes false before the data cell outputs. This gating takes place on the paper-tape controller logic card in the computer.

The reader is mounted to the front base panel to the left of the center of the panel.

A fanfold punched paper tape supply bin is mounted on the front panel to the left of the read station and a mirror image take-up bin is mounted on the front panel to the right of the read station. This reader take-up bin is between the read station and the take-up bin used for receiving tape from the punch.

Guides for reel supply and take-up of paper tape for the reader are as options to the basic fanfold means of handling tape.

Power Control

Power supply is controlled by a semiconductor TRIAC device. This device provides TTL logic level control of 50 or 60 Hz power at 110 or 220 Vac.

Punch power is normally off and is turned on by a demand from the computer to record data. Upon receiving the demand, the

punch power is automatically turned on and the punch speed is checked electronically. When the punch attains proper operating speed, the data in the buffer is recorded and a punch complete signal is sent back to the CPU. If the punch receives no data-record demand from the CPU for a period of two to four seconds, the punch power automatically turns off and waits for the next demand to record.

Reader power is controlled by a switch on the front panel. Idle running of the reader presents no major wear problem, compared to that created by idle running of the punch.

Interface Connections

Power for the paper tape punch/reader chassis can be 50 or 60 Hz cycle, 110 or 220 Vac, as specified by the user. Standard is 60 Hz at 110 Vac.

Power to the chassis is via a single ac power cord for both the reader and punch.

Logic signals are at standard TTL levels of 0.0 to +0.5 V false and +2.4 V to +5.5 V true.

Signals to and from the DM158 controller card located in the computer main frame are routed to the punch/reader chassis via a single cable with both ends having two connectors each.

Punch/Reader Combinations

A Remex Model RR-302 RB (300 character per second) unidirectional reader is the

paper-tape punch/reader

standard reader used in the paper-tape subsystem. The paper-tape punch may be, however, any one of the following:

1. Roytron 500 (50 character per second).
2. Roytron 700 (75 character per second).

3. Teletype BRPE11 (110 character per second).

The configuration of the Remex/Roytron combinations is shown in Figure 13-2. The configuration for the Remex/Teletype version is shown in Figure 13-3.

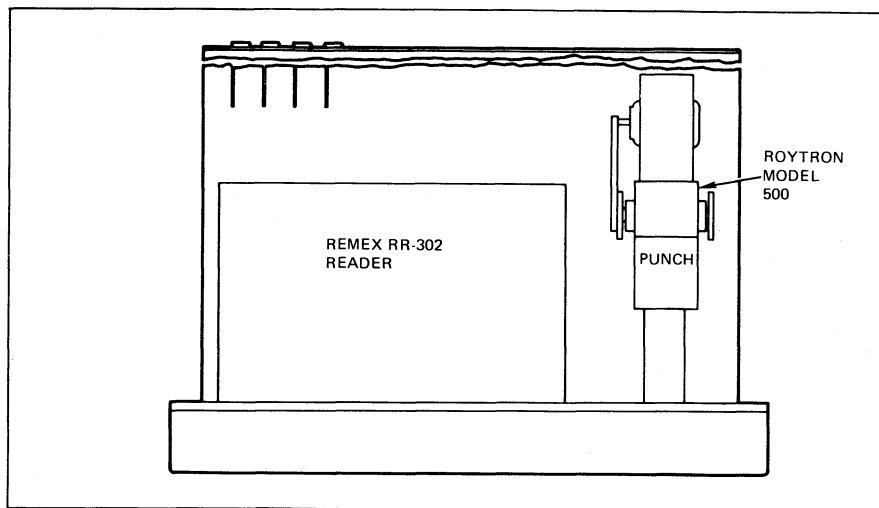


Figure 13-2.

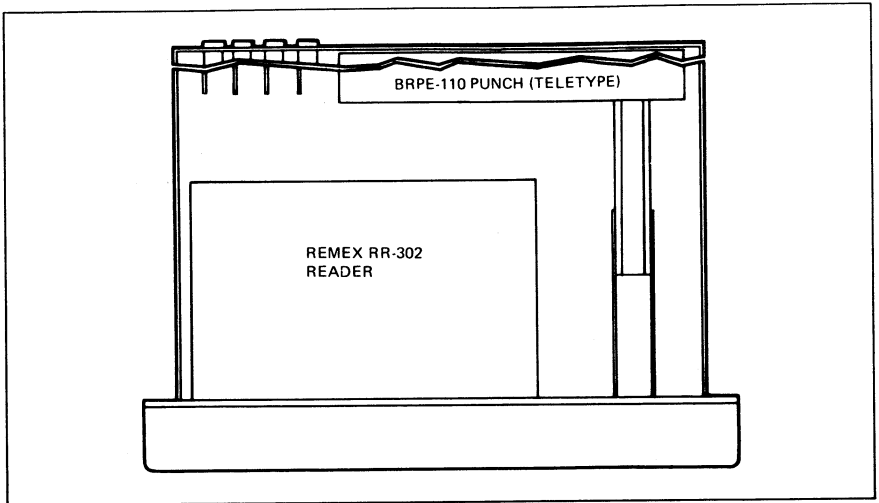


Figure 13-3.

SECTION 14 – OPERATION

Very little preparation of the Varian 520/i is required before running an object program. Assuming that the system, including peripherals, is properly installed and connected to a source of ac power, perform the following steps:

1. Press the power ON switch on the control panel.
2. Manually load the Basic Bootstrap program from the control panel (Figure 14-1).
3. From the teletype punched-tape reader, load the Binary Loader program (Figure 14-2).
4. From the teletype punched-tape reader, load the required binary object program or programs (Figure 14-3).
5. Press the RUN switch on the control panel.

The only decision that must be made in advance by the operator is the 1024-byte memory sector in which he wishes to load each program. In the figures that summarize the Bootstrap, Binary Loader, and object-program loading sequences, the S character represents the memory-sector code selected by the operator in accordance with the following:

<u>Memory Sector</u>	<u>S Character Code (Hexadecimal)</u>
0	03
1	07
2	0B
3	0F
4	13
etc.	etc.

The loading sequences summarized in Figures 14-1, 14-2, and 14-3 deal primarily with controls located on the computer front panel.

Program Assembly

The binary object program referred to in Step 4 above is initially prepared by writing the program in symbolic language and then processing it through the computer, using the Varian 520/i Assembler described in detail in Section 16. This process is summarized in Figures 14-4, 14-5, 14-6, and 14-7.

Checking the Program

After each program has been loaded, it may be desirable to examine the sequential memory locations that contain the program (or any other block of sequentially stored data).

operation

The STEP switch on the front panel of the computer can be used to accomplish this

objective by following the procedure summarized in Figure 14-8.

Manually Load 14 Memory Bytes	
Set MA Register	Set M Register
SA1	23
SA2	33
SA3	33
SA4	C1
SA5	32
SA6	41
SA7	F3
SA8	A5
SA9	31
SAA	01
SAB	79
SAC	00
SAD	F3
SAE	A1

Figure 14-1. Basic Bootstrap Program Loading Steps

Step	Description
1	Set P register to SA1.
2	Set X register to S00.
3	Position binary loader tape with first character over read station.
4	Press RESET switch.
5	Press RUN switch.
6	Verify correct loading by a computer halt with P register set to SBO.

Figure 14-2. Binary Loader Program Loading Steps

Step	Description
1	Set Q register to SF4.
2	Set B register to 0000 to load and halt; to a positive value to load and execute.
3	Set P register to SB0.
4	Position object tape blank leader over read station.
5	Press RESET switch.
6	Press RUN switch.
7	Verify that computer does not halt with P register set to SE2 due to a check-sum error. If a check-sum error occurs, perform steps 8 and 9.
8	Back-up the tape at least 13 inches and position a visual aid over the read station.
9	Press RUN switch.

Figure 14-3. Binary Object Program Operating Steps

Field	Coding Example	Coding Form Columns	Rules
Label	1	1-4	First character must be alphabetic. First two-character combination must be unique for each label.
Operation	2	8-11	Valid mnemonics are listed in appendix D. Operation code addition N may be used with skip-instruction mnemonics only. Operation code additions I and X may be used with memory-reference instructions only. Operation code additions always occur in column 4.
Variable	3	16-58	Absolute operands may be octal, decimal or hexadecimal single-valued, positive integers. First digit of octal value is zero. First digit of decimal value is non zero. Hexadecimal value is prefixed with \$. Literal operands may be any form of data and must be prefixed with =. Symbol operands may be any valid label.

Figure 14-4. Source Statement Coding Rules (Sheet 1 of 2)

Field	Coding Example	Coding Form Columns	Rules
Comment	4	16-58	May begin in second column after an operand. May begin in second column after the last value in a data-definition statement. May begin in column 16 if there is no operand.
	5	2-58	May begin in column 2 if * is placed in column 1.

Figure 14-4. Source Statement Coding Rules (Sheet 2 of 2)

Action	By	Description
1	Programmer	Set P register to \$01.
2	Programmer	Push down all SENSE switches. To punch source tape, go to action 3, otherwise go to 7.
3	Programmer	Turn mode knob to LOCAL.
4	Programmer	Turn on punch
5	Programmer	Press HERE IS key.
6	Punch	Blank tape leader feeds.
7	Programmer	Turn off punch.
8	Programmer	Turn mode knob to LINE.
9	Programmer	Press RUN switch.
10	Teletypewriter	Carriage return and two line feeds.

Figure 14-5. Preparation for Assembly

Action	By	Description
1	Programmer	Type a source statement one field after another. Follow each field with a space. Terminate the statement with a carriage return and line feed.
2	Teletypewriter	Carriage return and line feed if the statement is accepted. Error character is printed with carriage return and two line feeds if the statement is rejected. To correct a statement, go to action 3, otherwise go to 5 to punch the source statement or 10 to continue typing.
3	Programmer	Correctly retype statement.
4	Teletypewriter	Carriage return and line feed. To type without punching source tape go to action 10.
5	Programmer	Turn on punch.
6	Programmer	Type any character.
7	Punch	Punches source statement.
8	Teletypewriter	Prints source statement.
9	Programmer	Turn off punch.
10	Programmer	Input remaining source statements as described above.
11	Programmer	End input by typing an END or MOR statement.

Figure 14-6. Pass 1 Operation (Sheet 1 of 3)

Action	By	Description
		An END statement prepares the assembly program for pass 2 or 3; go to action 20. A MOR statement allows further input from punched tape or keyboard. For further keyboard input, go to action 12. For further input from punched tape, go to action 16.
12	Programmer	Press RUN switch.
13	Teletypewriter	Carriage return and line feed.
14	Programmer	Input remaining source statements.
15	Programmer	Go to action 11.
16	Programmer	Position blank source tape leader over read station.
17	Programmer	Set C register to \$C1.
18	Programmer	Press RUN switch.
19	Reader	Tape is read until an END or MOR statement is encountered. For an END statement, go to action 22. For a MOR statement with further keyboard input, go to action 12; with further tape input, go to action 16.
20	Programmer	Turn off punch.
21	Programmer	Input a carriage return and line feed.

Figure 14-6. Pass 1 Operation (Sheet 2 of 3)

Action	By	Description
22	Computer	Halts with P register set to \$7A1. To generate eight inches of blank tape, go to action 16, otherwise go to action 23.
23	Programmer	Press RUN switch.
24	Computer	Halts with P register set to 1.
25	Programmer	Go to Figure 14-7.
26	Programmer	Raise SENSE switch 2.
27	Programmer	Press RUN switch.
28	Programmer	Turn on punch.
29	Programmer	Type any character.
30	Punch	Outputs eight inches of blank tape.
31	Computer	Halts with P register set to 1.
32	Programmer	Go to Figure 14-7.

Figure 14-6. Pass 1 Operation (Sheet 3 of 3)

Action	By	Description
1	Programmer	Raise all SENSE switches.
2	Programmer	Position blank source tape leader over read station. For pass 2, go to action 3. For pass 3 go to action 8.
3	Programmer	Push down SENSE switch 3.
4	Programmer	Turn on punch.
5	Programmer	Press RUN switch.
6	Punch	Punches object tape.
7	Computer	Halts with P register set to \$7A1. For further passes, go to action 1.
8	Programmer	Push down SENSE switch 2.
9	Programmer	Press RUN switch.
10	Teletypewriter	Prints assembly listing of object record, source record and errors.
11	Computer	Halts with P register set to \$7A1. For further passes, go to action 1.

Figure 14-7. Pass 2 and 3 Operation

Steps	Description
1	Press down the P/Q REGISTER SELECT switch.
2	Press the CLEAR DISPLAY switch and enter the desired initial memory address into the switch register with the data entry switches.
3	Press the ENTER switch.
4	Press down the MA REGISTER SELECT switch.
5	Press the ENTER switch.
6	Press down the M REGISTER SELECT switch.
7	Press the DISPLAY switch. The data display indicators now show the contents of the desired address. The data may be altered at this time by performing steps 8 and 9, otherwise go to step 10.
8	Press the CLEAR DISPLAY switch and enter the desired data into the switch register with the data entry switches.
9	Press the ENTER switch.
10	Press the STEP switch. The data display indicators now show the next-highest sequential memory location to be accessed.
11	Press the DISPLAY switch. The data display indicators now show the contents of the address displayed in step 10. Repeat steps 8 through 11 as required until all desired addresses have been examined.

Figure 14-8. Sequential Memory Access Steps

SECTION 15 – BASIC BOOTSTRAP AND BINARY LOADER

Identification No. 92A0303-001A

Features

- Bootstrap contained in 14 bytes of memory; binary loader occupies adjacent 80 bytes of memory
- Bootstrap and loader relocatable to the top of any 1024-byte sector of memory
- Checksum computed for each record with halt on error
- Optional halt or transfer to program at end of load

The Basic Bootstrap is a program designed to load the Binary Loader into core and transfer program control to the Loader. The Binary Loader is designed to load into memory a formatted program object tape generated by the Varian 520/i Symbolic Assembler, Version A, or by a paper tape memory dump.

Since the Basic Bootstrap must be manually entered into the computer via the console, the Bootstrap is designed to be as small as possible. The Bootstrap is entered near the top of any memory sector and reads in the Binary Loader until it is overlaid, at which time program control is transferred to the Binary Loader and the computer halts.

The Basic Bootstrap requires 14 bytes entered manually, plus 1 additional byte which is loaded, then executed. The 15 bytes may be overlaid by the user's program after the Binary Loader has been entered.

The Binary Loader is 80 bytes long and normally occupies the uppermost part of core. The Loader is designed to load Symbolic Assembler object programs from a Model 33TC Teletype and will function with any computer memory size configuration.

Loading the Basic Bootstrap

Figure 15-1 details the 14 bytes (Location and Code columns) that make up the Basic Bootstrap program. The Comments column explains the function of each instruction.

The program is loaded into memory by successive console entries. The normal location for the program is the top of the memory. However, it may be loaded into the top of any sector (see Figure 15-2) by selecting one of the following values for S in the Location column, Figure 12-1.

<u>Sector</u>	<u>S</u>
0	\$3
1	7
2	8
3	F
4, etc.	13, etc.

basic bootstrap and binary loader

Location	Code	Symbol	Comments
\$ SA1	2333	A DXX	Decrement Index Register
SA3	33C1	FUN \$C1	Read one byte from tape
SA5	3241	SEN \$41	Sense reader ready
SA7	F3A5	BRU * -2	If not ready — Return to sense
SA9	3101	BTI	Input one byte
SAB	7900	STOX \$100	Store byte in memory, indexed
SAD	F3A1	BRU A	Return to A — get next byte

Figure 15-1. Basic Bootstrap Program

Loading the Binary Loader

The Binary Loader object tape is loaded by executing the Basic Bootstrap program as follows:

1. Set the X Register to \$S00 (value for S selected as indicated above).
2. Set the P Register to \$SA1.
3. Set the Binary Loader object tape in the tape reader with the first byte directly over the reader head.
4. Depress System RESET.
5. Depress RUN.

The tape is read into the computer and, upon completion, the reader and computer

halts and the P register should contain \$SB0.

The Bootstrap loads the Binary Loader backwards from the top of the sector until the loader overlays the Bootstrap and alters a branch command to transfer control to the top of the loader. An instruction is then executed which halts the computer. At this point, the Loader is ready to be operated and the instructions previously executed (locations \$SA1 to \$SAF) may be overlaid by the user.

Operating the Binary Loader

The Binary Loader is used to load a program object tape in the following manner:

1. Set the Q Register to \$SF4
2. Set the P Register to \$SB0
3. Set the B Register to zero or non-zero, according to the mode desired:

Zero Mode causes the Loader to halt the computer after the object tape is loaded.

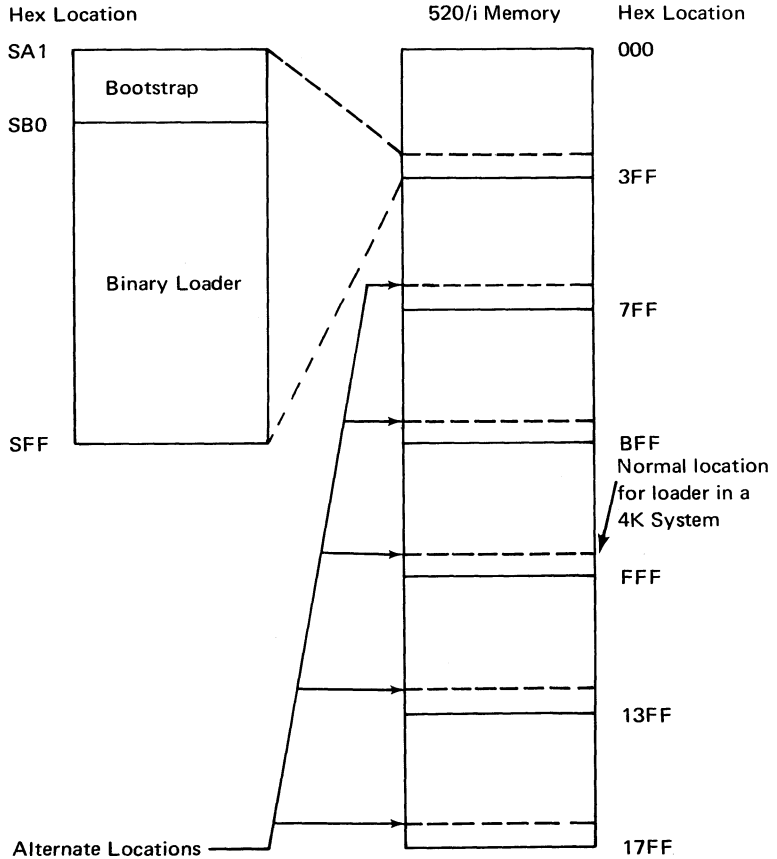


Figure 15-2. Basic Bootstrap and Binary Loader Memory Map

basic bootstrap and binary loader

Non-Zero Mode causes the Loader to branch to the transfer address on the object tape at the end of the load.

4. Set the program object tape in the tape reader anywhere on blank leader.
5. Depress System RESET.
6. Depress RUN.

The Binary Loader computes the checksum of each record when the length counter is zero and compares the computed value with the value on the object tape. If the two values are identical, the next record is input. If the two values are not identical, the

Loader halts at location \$SE2 indicating a checksum error.

A checksum error is caused by either an error in the object tape or an error in the data input process. A worst case error is an incorrect record length which causes the Loader to input as much as 127 bytes of data before checking the checksum.

To recover from a checksum halt, back up the object tape at least 130 bytes (13 inches) to a visual aid group (see Binary Tape Format, page A-34) and then depress RUN. If the checksum error continues to cause a halt at the same location, the object tape is incorrectly punched.

SECTION 16 – SYMBOLIC ASSEMBLER (Version A, Absolute Object) Identification No. 92A0303-002

Features:

- All symbolic, two or three pass assembly program.
- Interactive free-form keyboard input with on-line error analysis.
- Optional object tape output or listing under sense switch control.
- Segmentation of source program on separate tape segments.
- Dynamic assignment of location counter for each source tape segment.
- Restart capability on any pass at any time during assembly.
- Operands may be symbolic, octal, decimal hexadecimal, signed, modified symbolic, or character constants.
- Assembly listing includes paging, side by side object code and source statement output, and error flags.
- Indirect memory references supplied automatically when necessary.
- Eight pseudo operations and four data definition codes.
- 160 bytes of literal definition and up to 60 symbols per assembly in 4096 bytes

bytes of core expandable for larger core sizes.

The Varian 520/i Symbolic Assembler (Version A, Absolute Object) assembles source statements written in symbolic language and produces an assembly listing and object tape in loadable form for execution by the Varian 520/i processor.

The Assembler is designed as a two or three pass assembly program. Symbolic source statements which provide the input to the first pass, Pass I, must be re-input for each of the subsequent passes. Either Pass II (object code output) or Pass III (listing), or successive executions of Passes II or III, may be made following a single execution of Pass I.

Symbolic source statements are input from either the teletype paper tape reader or the teletype keyboard. Keyboard input applies only to Pass I; the input for Passes II and III must be on paper tape.

The Assembler identifies errors in the source input statements with appropriate flags on the assembly listing. When applicable, NOP (Non-Operative) instructions are assembled into the output object code in order to maintain continuity and provide convenience for patching without reassembly. Multiply-defined symbols are referenced to the location of the first occurrence of the symbol.

The location counter and machine language translations of the symbolic source statements appear in hexadecimal notation on the assembly listing.

Pass I generates a symbol table; assigns absolute values to all symbols; checks for illegal operation codes, undefined and multiply-defined labels; makes literal assignments and block storage assignments; and signals the end of pass.

Pass II formulates the binary object code from the source code, generates indirect address references when instruction memory references conflict with the hardware core sectorization, and punches the binary object code onto paper tape for subsequent loading into the processor.

Pass III lists the assembly output side by side with the input source statements and any error flags generated on the teletype-writer.

The user selects Pass I, II, or III by positioning the three sense switches on the console. For Pass I, all three sense switches are in the SET position. For Pass II, Sense Switch 3 is placed in the SET position. For Pass III, Sense Switch 2 is placed in the SET position.

It is recommended that in the early stages of writing a program, the Pass III listing be obtained in advance of a Pass II object tape in order to detect assembly errors more readily.

The Assembler is fully contained within a single loadable module on one paper tape. It is fully operable on a computer equipped with 4K of memory and an ASR 33 Teletype unit.

The Assembler Program is organized so that the literal pool and the symbol table follow the coded portion of the program. Because of the statement by statement processing of the source data by the Assembler Program, the object programs that are generated may extend to any size so long as the number of literals and symbols do not exceed the specified limits. A source program assembled on a 4K processor is restricted to a maximum of 60 symbols. Larger core sizes allow a greater number of symbols.

Pass II and Pass III of an assembly require paper-tape input. In the case of Pass I, however, the user has a choice of paper tape or keyboard. Moreover, by the proper use of the MOR Pseudo Operation (see below) a user may change, at will, his mode of input during Pass I from paper tape to keyboard and back. The flexibility of the Assembler input/output, plus the MOR Pseudo Operation, allows the user this option. The Assembler also provides automatic error diagnostics, on a statement by statement basis, for statements entered through the teletype keyboard. Only error free data is retained.

As data is received from the keyboard, it is arranged internally to agree with the standard symbolic coding format i.e., label beginning in position 1, operation code in position 8 and the operand in position 16 of the source input buffer. Data received beyond position 58 of the input buffer is not retained for output. Upon receiving a space character from the keyboard, the assembler advances to the first position of next field of input. The first non-space character from the keyboard marks the beginning of the next field. A carriage

return followed by a line feed character terminates an input statement. The assembler examines each statement for erroneous content. If the statement is free of error, the contents of the source input buffer, through position 58, are punched onto paper tape.

In the paper-tape input mode for Pass I, no on-line error diagnostics are provided to the user. However, the punched source statements produced during a keyboard input are accepted as a Pass I paper-tape input.

Preparation of Source Program

The source program is written as a series of symbolic instructions containing the following fields as shown in Figure 16-2:

Symbolic Label Field: One to four alphanumeric characters beginning in column 1. The first character must be alphabetic; the second character may be alphabetic or numeric. The first 2 characters of each label must be unique.

Operation Code Field: Any 3 character mnemonic value defined in the table of 520/i instructions (see Section 13).

Operation Code Appendage Field: Any one of the following alphabetic or numeric indicators:

I	Indirect Addressing
X	Indexed Addressing
N	Negate Skip Operation
1-4	Operand Precision for Literal Definitions

The operation-code appendage immediately follows the operation code. The appendage I, X and the numeric precision designators are appended only to memory-referencing codes. The N appendage may be appended to single-byte non-memory-referencing "skip" instructions.

Variable Field: Single-value operand or symbol. Operands can be symbolic, absolute or literal form. Absolute operands may be octal (first digit zero), decimal (first digit non-zero) or hexadecimal (prefixed with \$). Absolute operands must be positive integers. Literal definitions must be preceded by an equal (=) sign and may consist of any valid form of data. A symbol may be any valid label designation.

The Assembler allows a user up to four literal pools in one source program. Each pool has a capacity of 40 bytes with a maximum of four symbolic literal definitions per pool; thus a single source program may contain as many as 160 bytes of literal definitions including up to 16 symbolic literal definitions.

Two kinds of relative addressing are permissible:

- (a) relative to a symbol; e.g., $A + 5$, $A - 5$
- (b) relative to the current instruction; e.g., $* + 1$, $* - 1$

The basic unit for address incrementing and decrementing is the 8-bit. The incrementing or decrementing factor must be an octal, decimal or hexadecimal integer.

Comment Field: Begins following the first blank character after the operand or following the first blank character after the last

symbolic assembler (version A)

data value of a data definition statement. In the absence of an operand, a comment may begin in column 16. An asterisk (*) in column one will identify the entire statement as a comment.

Different types of data may appear in a single Data Definition statement. In the case of an alphanumeric constant contained in a data definition statement, the precision is ignored and is limited in extent only by the size of the source input buffer (which does not accept data beyond column 58 of the operand field).

Multiple operands in a data statement must be separated by commas and a blank character must follow the last piece of data within a given data statement. Numeric values may be signed or unsigned. The Assembler Program generates zeros to the precision specified for each successive comma contained in the operand field of the data definition statement and for the blank or end-of-line character following a comma at the end of a data definition statement.

All values are right justified within the specified number of bytes. Negative integers are represented in 2's complement form. Whenever an alphanumeric constant is not evenly divisible by the specified precision, the odd remaining bytes are left-adjusted and padded with leading binary zeros.

Assembler Pseudo Operations

The following pseudo operations are recognized by the Assembler:

BSS	Block Storage Specification
ORG	Set Location Counter

SET	Set Symbolic Value
EQU	Equate Symbols
END	End Assembly
LIT	Begin Literal Block
MOR	Halt for More Input
IOR	Begin Indirect Pointer Block

The precision of a constant in storage is defined by one of the following pseudo instructions:

DA1	(for single or multiple 1 byte definition)
DA2	(for single or multiple 2 byte definitions)
DA3	(for single or multiple 3 byte definitions)
DA4	(for single or multiple 4 byte definitions)

The Assembler interprets the mnemonics for these pseudo operations as programmed directives to itself and no object program data is directly generated.

The same is true of data definition codes, DA1, DA2, DA3, and DA4, which are used to define single or multiple constants in storage.

The following paragraphs present detailed descriptions and operating information for each pseudo operation and data definition code.

BSS Block Storage Specification

This instruction reserves a block of memory locations (without affecting the contents)

and assigns a name to the block. The size of the block is defined by the numeric quantity in the variable field. Memory locations in the block are sequential. The label refers to the location with the lowest address. Successive locations are referred to as label +1, label +2, etc.

Coding Sample:

BL	BSS	\$5A	BUFFER A
HE	BSS	0100	BUFFER B

ORG Set Location Counter

This instruction alters the location counter to the value specified in the variable field. A symbol in the variable field must have been previously defined; that is, it must have appeared in the label field of some instruction previous to this instruction. A negative or undefined quantity in the variable field is ignored and causes no change to the program counter. A symbol in the label field is assigned the value in the variable field. Subsequent object code will be assigned addresses relative to this new definition of the location counter.

Coding Sample:

SA	ORG	\$100	SET LOCATION COUNTER TO \$100
	ORG	JO	SET LOCATION COUNTER TO THE VALUE OF JO
	ORG	99	SET LOCATION COUNTER TO 99

SET Set Symbolic Value

This instruction changes the address value of the symbol in the label field to the value in the variable field, overriding any previous value assigned to the symbol. A symbol in the variable field must have been previously defined; otherwise, the instruction will be flagged as undefined and the symbol in the label field will be assigned the value of 0.

Coding Sample:

JOE	SET	\$3FC	ASSIGN JOE THE ADDRESS VALUE 3FC (HEX.)
LIST	SET	ALFA	ASSIGN LIST THE SAME ADDRESS VALUE AS ALFA

EQU Equate Symbols

This instruction assigns the symbol in the label field an address value equal to the value in the variable field. If the symbol was previously defined, the instruction will be flagged as multiply-defined and the symbol will retain its previous value. A symbol in the variable field must have been previously defined; otherwise, the instruction will be flagged as undefined and the symbol in the label field will be assigned the value of 0.

Coding Sample:

RATE	EQU	JOE	ASSIGN RATE THE SAME ADDRESS VALUE AS JOE
------	-----	-----	---

```
SAM    EQU    *+$10  ASSIGN SAM
                        AN ADDRESS
                        VALUE OF
                        10 (HEX.)
                        GREATER
                        THAN
                        CURRENT
                        LOCATION
                        COUNTER
```

be assigned locations beginning with the current location counter value. If a LIT instruction is not included in the source program, all literals in the program are assigned locations starting with the first location after the last address in the program.

Coding Sample:

```
LIT    *+$80  ASSIGN LITERAL
                        ADDRESSES BEGINNING
                        AT THE CURRENT
                        LOCATION COUNTER
                        PLUS 80 (HEXA
                        DECIMAL)
```

END End Assembly

This instruction terminates assembly of the program and must be the last statement of a source program. As a result of this instruction, the program counter is set to the value in the variable field when the object program is loaded into memory. An undefined symbol, invalid quantity, or a blank in the variable field causes the program counter to be reset to zero after loading is complete.

Coding Sample:

```
END    BEGN    END ASSEMBLY
                        BEGIN EXECUTING
                        OBJECT PROGRAM
                        AT LOCATION
                        BEGN
```

MOR Halt for More Input

This instruction provides the programmer control of the program input mode. When the Assembler encounters a MOR instruction, the computer halts at location \$61B.

If further input is to be made from the keyboard, the programmer must press the RUN switch. The Assembler then issues a carriage return followed by two line feeds to the teletype and the programmer may begin typing source statements.

LIT Begin Literal Block

This instruction reserves a block of memory locations for all literals that have occurred since the previous LIT instruction or since the start of the program. The value in the variable field specifies the lowest address of the block. A symbol in the variable field must have been previously defined. An undefined symbol, invalid quantity or a blank in the variable field, causes literals to

If further input is to be made from punched tape, the programmer must load the source program tape to some part of the blank leader preceding the first statement, manually enter the value \$C1 into the C register of the processor, and press the RUN switch twice. (\$C1 identifies the teletype paper tape reader as an input device.)

Coding Sample:

```
MOR    HALT ASSEMBLER
```

IOR Begin Indirect Pointer Block

This instruction reserves a block of memory locations for indirect addresses in the object program. The value in the variable field specifies the lowest address of the block and must not be larger than \$3FA if the block is to accommodate the maximum number (3) of indirect addresses within the lowest (zero) sector of the object program. The IOR instruction must appear in the source program before any indirect addresses are generated.

If the programmer includes an IOR instruction in the source program, the assembly program generates an indirect memory reference whenever an addressing function cannot be directly resolved. The Assembler then inserts a reference to the indirect pointer block in the object instruction along with an indirect-through-zero-sector addressing code. The actual indirect memory address generated by the Assembler is stored in the indirect pointer block at the reference location inserted in the object instruction. The Assembler prints a I on the assembly listing to the right of the operation code of the source instruction for which the indirect address is generated.

Coding Sample:

```
IOR  $20  RESERVE 8-BYTES OF
        STORAGE FOR INDIRECT
        POINTERS BEGINNING AT
        LOCATION 20 (HEXA-
        DECIMAL)
```

DA1, DA2, DA3, DA4, Define Constants

These instructions define data constants to be constructed by the assembler. The pre-

cision of each defined data word is the last character of the instruction mnemonic. For example, DA3 specifies data to a precision of three bytes (24 bits).

This instruction may specify one or a series of constants. A data constant is any valid octal, decimal, or hexadecimal integer. An alpha constant is any series of alphanumeric characters enclosed between two prime (') characters. (A double prime is interpreted as a single prime when it is part of an alphanumeric constant.) An address constant is any valid address symbol. Multiple constants in a data definition statement must be separated by commas.

More than one type of constant may be included in a data definition statement.

A symbol in the label field is given the address of the left-most byte of the variable field. Each constant value in the variable field, except an alphanumeric constant, is allotted the number of bytes of storage specified by the last character of the instruction mnemonic. The constants are right justified (toward the highest address) within the allotted locations. Alphanumeric constants are allotted one byte of storage for each character. If the total storage block allotted for a data definition instruction is not an even multiple of the specified precision, bytes of zeros are stored to fill the difference.

The specified number of bytes, filled with zeros are allotted for invalid data constants. Binary zeros will be generated in storage to the specified precision for each null field. A null field may be indicated by adjacent commas or by a blank or carriage return following the last comma in the variable field.

symbolic assembler (version A)

Coding Sample:

CON	DA1	1,2,3,, 0100,\$FF	DEFINE SINGLE BYTE CON- STANTS AT CON, CON+1,ETC.
EMSG	DA2	12, 'ERROR STOP', \$8D8A	DEFINE ERROR TYPE — OUT

Assembler Output

During Pass I, the Assembler generates a symbol table, assigns absolute values to all symbols, and makes literal assignments, and block-storage assignments. Pass I produces a symbolic tape carrying all of this source-program information.

During Pass II, a binary object program is formulated from this data and punched onto paper tape in the standard binary tape format (see page A-34).

During Pass III, the binary object program is typed side-by-side with the source statements on the teletypewriter. The format of this Assembly Listing is shown in Figure 16-3 and its component elements described in the following paragraphs.

Location Counter (columns 1-4): Four digit hexadecimal integer specifying the memory address of the most significant byte of the immediate machine instruction or data quantity (see Assembler Address section below).

Object Code (columns 7-10): Hexadecimal representation of one-byte or two-byte

machine language instruction, memory address value or data quantity.

Error Flag (column 12): Error code identifying the most severe errors in the source statement:

Error Flag	Definition
O	Invalid operation mnemonic
U	Undefined symbol
M	Multiply-defined symbol
P	Invalid operand precision specified
N	Invalid numerical operand
I	Invalid operation code appendage
A	Addressing error

Source Statement (columns 14-72): Input source statement.

Assembler Addressing

The addressing mode is specified, in the Varian 520/i, through bits 10, 11, and 12 of the machine instruction word in accordance with the following table:

Bit 12	Bit 11	Bit 10	
0	0	0	} Direct to lower 4 sectors
0	0	1	
0	1	0	
0	1	1	
1	0	0	Direct to current sector

Bit 12	Bit 11	Bit 10	
1	0	1	Indirect through current sector
1	1	0	Indexed
1	1	1	Indirect through sector 0

The memory of the 520/i is divided logically into 1024-byte sectors. Addresses which are direct and which fall inside the current sector are specified as being direct-to-current-sector without exception. When a user specifies Indirect Addressing by use of the operation code appendage, the Assembler automatically sets the proper mode bits provided that:

1. The operand address references a location inside the 0 sector.
2. The operand address references a location inside the current sector.

In the event that the current sector is also the 0 sector, the indirect-through-current-sector mode will be selected.

In the case of an addressing infraction (e.g., when an operand address directly references a core location which is outside of the current sector and which is above core location 4096, or when an operand address indirectly references a core location which is not inside the current or 0 sector), the Assembler takes one of two actions:

1. Inserts the letter A in the error flag field of the assembly listing and substitutes two NOP's for the instruction object code.

2. Relocates the operand address to a user designated area in the 0 sector and generates the required indirect reference to that address in the object code for the instruction.

Action 2 will be selected provided that the user has included a valid IOR statement (Indirect Pointer Block Pseudo Operation) in his source program. The letter I will appear in the column following the operation code appendage on the assembly listing as a visual indication to the user.

Assembler Operation

After the Assembler has been loaded into memory, the user should take the following preliminary actions prior to beginning his assembly.

1. Set the P register to 1.
2. Push down all SENSE switches on the Varian 520/i console.
3. Turn the teletype punch OFF.

If the source program is to be input through the paper tape reader, the user should proceed from Step 28 below.

For keyboard input where a source program tape is to be created:

4. Turn the teletype mode knob to the LOCAL position.
5. Turn the teletype punch ON.
6. Press the HERE IS key to feed out blank leader at the start of the source tape.

7. Turn the teletype punch OFF.
8. Turn the teletype mode knob to the LINE position.
9. Press the RUN switch on the Varian 520/i console.

The Assembler issues a carriage return followed by two line feeds to the teletype and then waits for a keyboard input.

The following procedure will allow the user to enter assembly language source statements from the teletype keyboard and, at the same time, to punch these statements in fixed format onto paper tape through the teletype paper tape punch:

10. Type a source statement.
11. Terminate the statement with a carriage return followed by a line feed.

(The user may cancel a statement at any time prior to the carriage return and line feed by typing a ←. A new statement will begin with the next typed character.)

The Assembler does a complete analysis of the statement. If the statement contains an error, the Assembler types a single flag character indicating the type of error that is detected, rings the bell once and issues a carriage return followed by two line feeds to the teletype. The statement is not assembled. The user should re-enter his statement correctly, using the teletypewriter input.

If the statement is accepted, the Assembler rings the bell, issues a carriage return followed by a line feed to the teletype, and

waits for the following response from the user:

12. Turn the teletype punch ON.
13. Enter a single character from the keyboard (any character is valid).

The Assembler immediately prints the entire source statement in fixed format on the teletypewriter and, at the same time, punches the statement onto paper tape. (If the user does not want the statement to appear on his source program tape, he should not turn on the paper tape punch. In this case, the statement is only printed on the teletypewriter.) The Assembler issues a carriage return followed by a line feed to the teletype after printing the statement and waits for a further input from the teletype keyboard. The user should then:

14. Turn the paper tape punch OFF.
15. Type the next source statement.

After entering an END statement, the user should proceed as outlined above to the step where the Assembler is waiting for the next source statement. At this point, the user should replace Steps 14 and 15 with the following:

16. Turn the paper tape punch OFF.
17. Type a carriage return followed by a line feed.

This action will cause the Assembler to ring the bell twice and halt with the program counter set to \$7A1. The following action by the user will result in eight inches of

blank tape being output by the paper tape punch:

18. Raise Switch 2 to the RESET position.
19. Press the RUN switch.
20. Turn the paper tape punch ON.
21. Enter a single character from the keyboard (any character is valid).

After punching the eight inches of blank tape, the computer will halt with the P register set to 1. (If the user does not want any blank tape punched, he should press RUN when the halt occurs at \$7A1, and the computer will halt with the P register set to 1.) After the computer has halted with the P register set to 1, the user should:

22. Remove the source program tape from the punch head.
23. Set the sense switches for the desired assembler pass (raise Sense Switches 1 and 2 for Pass II, object paper tape, or raise Sense Switches 1 and 3 for Pass III, assembly listing).
24. Adjust the source program tape in the paper tape reader to some part of the blank leader that precedes the first statement of the source program.

If Pass II is to follow, the paper tape punch should be turned ON. To insure that blank leader precedes the first record of the object program tape, the user should perform Steps 3 through 8, above, before continuing with the following:

25. Press RUN to initiate the pass indicated by the sense switch settings.

Upon completion of the pass, the computer will again halt with the P register set to \$7A1. To continue with the next pass, turn the paper tape punch OFF and:

26. If the next pass is Pass II, the user should proceed from Step 18 above.
27. If the next pass is Pass III, the user should press the RUN switch and proceed from Step 22 above.

To begin a new assembly, the user should return to Step 1 and proceed from there.

If the source information to Pass I is contained on paper tape, the first statement to be entered from the keyboard must be a MOR statement. When the Assembler recognizes the MOR statement, it will ring the bell at the teletype and halt the computer with the P register set to \$61B. (The bell is not sounded when the MOR statement is encountered during Pass II because of the extraneous punches that would occur.) If further input, after the MOR statement, is to be made through the paper tape reader, the user should:

28. Adjust the source program tape in the paper tape reader to some part of the blank leader that precedes the first statement contained on the tape.
29. Manually enter \$C1 into the least significant byte of the C register.
30. Press the RUN switch. (Unpredictable results can occur if the \$C1 value is not properly entered. If this occurs, the user should stop the computer, reset the P register to \$61B and enter the correct \$C1 value into the C register before pressing RUN.)

symbolic assembler (version A)

There will be no further interaction with the user until another MOR statement or an END statement is encountered.

If, after a MOR statement, further source statement inputs are to come from the keyboard (on Pass I), the user should press RUN following the halt at \$61B. The Assembler will print the MOR statement in fixed format, issue a carriage return followed by a line feed to the teletype and wait for the user to enter the next source statement from the keyboard.

Since the Assembler is initially in the keyboard mode for Pass I, the user may continue in this mode of input for several statements. At some point, however, he may wish to interrupt his program and relinquish the use of the computer to another user. If his program is incomplete, the first user should terminate this segment of his source program tape with a MOR statement.

At some future time, when he again has use of the computer, the first user can initiate the assembly process by loading the previously punched source program tape into the paper tape reader, entering a MOR statement and manually entering the \$C1 code in the C register. The Assembler will then proceed to read and process each of the statements contained on the source program tape for Pass I. When the Assembler encounters the MOR statement terminating the source tape segment, it will halt and wait for user action.

If the user elects to continue his program in the keyboard mode, he will simply press RUN and the Assembler will return him to the keyboard mode of input. If other source paper tape segments are to be included, the user will ready the next source program tape

in the paper tape reader, manually enter \$C1 into the C register and press RUN. The Assembler will proceed as described above by reading the source statements from the paper tape reader until another MOR statement is encountered or until an END statement is encountered.

The paper tape mode of input is assumed for Passes II and III of the Assembler; therefore, the statements that make up the user's source program must be on paper tape (in either fixed or free form) prior to beginning either of these passes. The user must adjust his source program tape in the paper tape reader to some part of the blank leader which precedes his initial source statement. He then presses the RUN switch to initiate the assembler pass specified by the console switches. For multiple paper tape segments where each segment is terminated by a MOR statement, the user must follow the procedure outlined above for the MOR statement in order to continue with the next segment of his source program. The pass is terminated by an END statement. The same procedure for the END statement applies for all passes.

In the course of an assembly, and particularly during either Pass II or Pass III, the Assembler may type out the letter S and halt with the P register set to 1. By this action, the Assembler has notified the user that:

1. His source program has too many symbols, or
2. His source statements, input on Pass II or III, are not synchronized with Pass I; that is, a statement label does not have the location value that it had during Pass I.

There is no direct recovery from either of these errors. For the first situation, the user must rework his source program (e.g., segment it or use more self-relative references) in order to reduce the number of symbols. In the second situation, simply restarting from the beginning of the pass should correct the problem since the malfunction

may be attributed to a misread source statement by the paper tape reader. If the problem persists, the user should be sure that the sequence of his source statements has not changed from that of Pass 1. If no discrepancy is apparent, he should run the teletype test program to determine if there is a hardware malfunction.

SECTION 17 – SYMBOLIC ASSEMBLER (Version B, Relocatable Object) Identification No. 92A0303-004

Features:

- All symbolic two or three pass assembly program.
- Symbols defined in one program can be referred to in another, thus affecting symbolic linkages between independently assembled programs.
- Object code produced permits unrestricted program relocation.
- Permits literal external symbolic references.
- Interactive free form keyboard input on first pass, or free form source tape input for all passes.
- Alternating between keyboard and paper tape input under program control.
- Source tape input for all passes from teletype reader or high speed paper tape reader.
- Complete error analysis and diagnostics performed on each statement input.
- Restart capability on any pass at any time during assembly.
- Pseudo operations include ENTRY and EXTERNAL symbol definitions.

The Varian 520/i Symbolic Assembler (Version B, Relocatable Object) assembles source statements written in symbolic machine language and produces an assembly listing and relocatable object program in loadable form for execution in the Varian 520/i processor.

The Assembler is designed as a 3-pass assembly program and is essentially identical to the Version A (Absolute Object) Assembler described in Section 16. The major difference is that Version B produces relocatable object code and allows symbols to be defined in one program and referred to in another, thus effecting symbolic linkages between independently assembled programs. This permits source programs written for the Version B Assembler to be relocated at load time from their originally assembled origins and placed in any other suitable core area, and also permits references to data and transfers of control between programs that have been assembled separately, but are loaded together.

Object programs produced by the Version B Assembler are loaded into the computer by using the Relocating and Linking Loader described in Section 18. (The Binary Loader, described in Section 15, loads only absolute object code format.)

The operation of Pass I of the Version B Assembler is essentially the same as that of Version A. During Pass II, however, the

symbolic assembler (version B, relocatable object)

Version B Assembler produces a binary object tape that includes relocatable object code and coded directives that will enable the Relocating and Linking Loader to perform its function. Pass III produces an Assembly Listing.

Since the Version B Assembler occupies additional space in memory, only 38 symbols are allowed per source program, and only a total of 100 bytes of literal definitions (see Figure 17-1).

Preparation of Source Program

Source Programs for the Version B Assembler are written in the same manner as those for Version A (Section 16) using the pseudo operations described below.

Assembler Pseudo Instructions

The Version B Assembler recognizes the following pseudo instructions:

BSS	Block Storage Specification
ORG	Set Location Counter
SET	Set Symbolic Value
EQU	Equate Symbols
END	End Assembly
LIT	Begin Literal Block
MOR	Halt for More Input
ENT	Define Entry Point Symbol
EXT	Define External Symbol

Note that the IOR instruction (Begin Indirect Pointer Block) recognized by the Version A Assembler is not included in this

list for the Version B Assembler. The IOR function is performed by the Relocating and Linking Loader used to load a Version B object program.

Two of the instructions in the above list are unique to the Version B Assembler. ENT and EXT allow symbols to be defined in one program and referred to in another, providing symbolic linkages between independently assembled programs. The linkages can be effected, however, only if the object tape provides information about the linkages to the Relocating and Linking Loader. The Loader resolves these linkage references at load time.

A linkage symbol is identified to the Assembler by means of the ENT (entry) instruction. This identifies the symbol as an entry point which may be referenced by another program in order to effect an external branch operation or a data reference.

Similarly, the program that references a symbol defined in some other program may identify it by means of the EXT (external) instruction. The ENT and EXT pseudo operations are described in detail in the following paragraphs.

ENT - Define Entry-Point Symbol

This instruction defines linkage symbols in this program which may be used by some other program.

The symbol in the Entry label field may be used as an operand in other programs. The symbol may also be used as an operand in the program itself. The variable field of the

ENT instruction is not processed by the Assembler.

An ENT instruction must appear in the source program at each point that the programmer wishes an entry point defined. The symbol in the label field will be treated as a local symbol as well as an entry symbol and refers to the following instruction; thus, the same symbol must not appear in the label field of another instruction in a program. The symbol which appears in the label field of the ENT instruction will be assigned an address value equal to the current value of the location counter.

EXT - Define External Symbol

This instruction identifies linkage symbols that are referred to by this program but are defined in some other program. Each external symbol must be identified. The external symbol must appear as the only entry in the variable field of the EXT statement. Any adjustment factor attached to the symbol will be ignored. The program must reference an external symbol by means of the symbol in the label field of the ENT statement in the external program. The symbol which appears in the variable field of the EXT statement may be used as a local symbol elsewhere within the program since the symbol identified in the EXT is not put in the symbol table for the program.

Assembler Output

The object tape produced by the Version B Assembler is in relocatable binary-tape format. The four frames following the block-start address (see page A-34) contain a 32-bit control record consisting of coded

directives to the loader. Also incorporated into the object data block is additional loader information, referred to as loader text, consisting of ASCII characters representing entry point and external symbols.

During Pass III, the Version B Assembler produces an Assembly Listing similar to that of Version A except that two additional error flags, L (Invalid Label) and E (External Symbol), are used to identify source statement errors. The complete table of Version B error flags is as follows:

<u>Error Flag</u>	<u>Definition</u>
O	Invalid operation mnemonic
U	Undefined symbol*
L	Invalid Label
M	Multiple defined symbol
D	Invalid Data Specification
P	Invalid operand precision specified
N	Invalid numerical operand
I	Invalid operation code appendage
A	Addressing error
S	Symbol table full or synchronization error
E	External symbol

*An undefined symbol in the variable field of a BRM (Branch and Mark) or CSQ (Change Status and Set Q Register) instruction or an undefined address constant in a literal operand will be processed in the same way that a symbol in the variable field of an EXT instruction is processed; that is, it will be treated as an external symbolic reference and will be flagged with the letter E.

Assembler Addressing

The addressing performed by the combination of the Version B Assembler and the Relocating and Linking Loader differs in part from that of the Version A Assembler and Binary Loader.

As in the case of the Version A Assembler, memory references in instructions which reference memory directly are specified in the direct-to-lower-four-sectors mode. Therefore, during an assembly the location counter for the object program as well as all references contained within the object program must not exceed the value of 4095. When an instruction contains an operand address which is larger than 4095, the instruction will be flagged and NOP's substituted in the object code.

Whenever applicable, indirect memory references within instructions are specified in the indirect-through-current-sector addressing mode. However, when the referenced indirect pointer does not lie in the current memory sector, the direct addressing mode is specified in the instruction word and the proper code is output to inform the Relocating and Linking Loader that the

instruction contains an unresolved indirect reference which the loader must resolve at load time.

Although at assembly time the object program is restricted to storage addresses which do not exceed 4095, the object program may be loaded into and executed from any part of memory and is restricted only by the physical size of the machine.

The Version B Assembler does not generate indirect pointers to resolve addressing infractions within the object program. The mechanism for generating indirect pointers is incorporated in the Relocating and Linking Loader.

Assembler Operation

The Version B Assembler is operated the same as the Version A Assembler with the following exceptions:

1. The END statement halts the computer with the program counter set to \$80C.
2. The MOR statement halts the computer with the program counter set to \$666.

SECTION 18 — RELOCATING AND LINKING LOADER

Identification No. 92A0203-007

Features:

- Allows programs to be relocated from their originally assembled origins
- Effects linkages among programs assembled independently but loaded together
- Loader itself is relocatable, and may be used as a callable subroutine to handle program overlays
- Optional listing of all symbolic references provided under sense switch control
- Loads from either a teletype or a high speed paper tape reader
- Detects multiply-defined symbols
- Alerts user when programs overlay loader or symbol table
- Detects tape checksum discrepancies

The Varian 520/i Relocating and Linking Loader operates on the binary object program produced on punched paper tape by the Varian 520/i Symbolic Assembler, Version B.

The Relocating and Linking Loader routine allows programs to be relocated at load time

from their originally assigned area to any other suitable area in the Varian 520/i memory.

The Loader also effects linkages between programs which are assembled independently but which are loaded together. It permits the user to combine separately assembled segments of a single program or to integrate independently assembled programs at load time into whatever memory configuration is suitable for a given computer run. The Loader implements all linkages indicated to effect transfer of control and referencing of data between programs.

The Loader may itself be used as a callable subroutine. Details on this version of the Loader are contained in the final pages of this section.

Loader Input

The Relocating and Linking Loader is input to memory using the Varian 520/i Binary Loader program (Section 18). The normal location of the Loader is shown in Figure 18-1.

The basic hardware requirements for the Relocating and Linking Loader are a 4096-byte Varian 520/i computer and a Model 33TC Teletype. The Loader requires 725 bytes of continuous memory beginning at location \$D2B and extending through

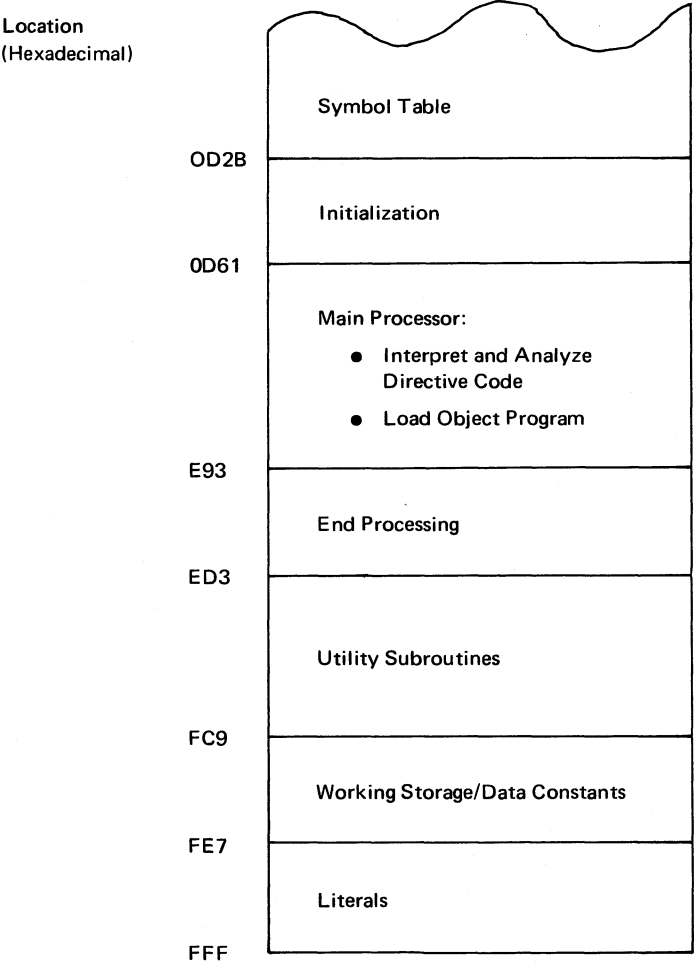


Figure 18-1. 520/i Relocating and Linking Loader Memory Map

location \$FFF. The Symbol Table is built downward in memory beginning with location \$D2A.

The Loader accepts input from either the Teletype paper tape reader or the high-speed paper tape reader attached to the 8-bit I/O adapter.

Object-Type Input

The punched object paper tape produced by the Symbolic Assembler, Version B, provides the normal input to the Loader.

The format of the standard Varian 520/i binary tape is described on page A- . An Assembler-assigned location address for the object code in the record is contained in the two bytes following the byte indicating the record length (there are a maximum of 31 bytes per record). This is followed by a four-byte Control Word containing binary coded directives to the Loader. The directives are arranged consecutively from left to right in the control word, corresponding to the data bytes that follow.

The remainder of the information in the record, with the exception of the last frame, consists of the binary object code and certain loader text in the form of ASCII-coded ENTRY and EXTERNAL symbols. The amount of information is determined by the record length and extends for as many bytes as are indicated by the count contained in the record length.

The loader text, together with the directive codes, enables the Loader to relocate the object code and to effect linkages between it and other load modules.

The last frame in the record contains the checksum as computed by the assembler.

Sense Switch Settings

The sense switches on the front panel of the computer have the following significance for the Loader:

- **Sense Switch 1**, when set, causes the Loader to halt after a module is loaded. When processing is resumed, the Loader proceeds according to the C register contents.
- **Sense Switch 2**, when set, causes the Loader to list the Symbol Table on the teletype after a module is loaded. Unresolved symbols are indicated by a high order one bit in the listed value.
- **Sense Switch 3** is not used by the Loader.

Operation of Loader

The Relocating and Linking Loader is initiated by entering the following parameters into the specified registers after the Loader is resident in core:

X Register: Relocation bias, or the amount by which the program is to be displaced at load time from its originally assigned area.

C Register: The beginning address of a table in the lower 1024-byte memory sector in which operand addresses are to be stored; these addresses are referenced indirectly through pointers inserted by

relocating and linking loader

the loader into memory-reference instructions which must use indirect addressing.

Q Register: Value \$F26 if the input device is to be the teletype paper tape reader, or value \$F37 if the input device is to be the high speed paper tape reader attached to the 8-bit I/O adapter.

P Register: Value \$D2D, or the starting address for the Loader.

After the specified values have been entered into the registers, and the Sense Switches have been properly set, the tape to be loaded is placed in the paper tape reader and positioned anywhere on the blank leader. The user then depresses the RUN button to execute the Loader.

On execution, the Loader saves the contents of the C register, marks the beginning of the symbol table, and proceeds as follows:

1. Transfers the contents of the X register to a 2-byte storage area reserved for the relocation bias.
2. Reads any blank leader which precedes the first Record Mark.
3. Reads Record Mark.
4. Reads the Record Length and retains the value in the Record Byte Counter (B register).
5. Initializes the beginning of the checksum by transferring the binary content of the first frame of the record into the lower-order 8 bits of the Y register. (Thereafter, as each frame is read, the 8-bit contents will be Exclusive OR'd with the

previous contents of the Y register and the results retained in the Y register as the current checksum.)

6. Tests the Record Length to determine if the end of the load module has been reached. The final record of a load module is identified by a Record Length of zero.
7. Reads the record-start address as assigned by the Assembler, adds the relocation bias, and retains the results in the X register as the initial value of the Location Counter.
8. Inputs the 4-byte Control Word into the 4-byte accumulator.

The Control Word contains the binary directive codes representing loader directives (see below). These are analyzed and interpreted by the Loader from left to right across the 32 bits of the Control Word, with a one-for-one correspondence between the directive codes and the data that follows on the object tape.

After each directive code has been interpreted and the associated object code processed and transferred to memory, the Loader performs the following "house-keeping" functions.

1. Increments the Location Counter by the size of the data which was transferred to memory in the preceding loading process.
2. Decrements the Record Byte Counter (B register) by the size of the data previously processed.

3. Compares the value of the Location Counter with the highest previous location counter value; retains the larger of the two values in storage reserved for the highest location counter value for use as the relocation bias for the next load module; and compares the latter value with the address of the next available space in the Symbol Table. If there is an overlap between the program being loaded and the Symbol Table, the Loader types an "X" on the teletypewriter, followed by a carriage return and line feed, and halts the processor at a prespecified address (\$E82).

4. Tests the Record Byte Counter (B register); if the contents are not 0, loads the Control Word into the 4-byte accumulator, left rotates the Control Word one binary position in order to right adjust the beginning of the next directive code in the 4-byte accumulator and repeats the above process.

5. If the Record Byte Counter (B register) is 0, transfers the relocation bias from storage to the X register. Reads the next frame from the object tape containing the assembler computed checksum of the data contained in the record and compares this value with the Loader computed checksum to verify the correct input of the data contained in the previous record. In the event of a discrepancy between the independently computed checksums, the Loader halts the processor at a prespecified location address (\$E91). (The halt action allows the user the opportunity to reposition the object tape to the beginning of the record mark of the record which was just processed.)

Upon completion of reading in a load module and verifying the checksum, the Loader proceeds as follows:

1. Tests Sense Switch 2 and, if set, lists all symbols and their values side by side on the teletypewriter before proceeding.

2. Inputs the next two frames from the object tape and transfers their contents into both the A and the C registers. These contents consist of an execution address for the program just loaded or all ones (—1) if an execution address was not specified at assembly time. A positive value in the C register is interpreted as a valid execution address and is offset by the relocation bias before the Loader proceeds with the next step.

3. Zeros the contents of the X register. The X register is used as a switch so that, should both Sense Switch 1 be set and the C register contents be negative, the Loader does not halt the computer twice before returning to load the next module.

4. Tests Sense Switch 1 and, if set, proceeds to step 5 below, otherwise tests for a positive value in the C register (an indication of a valid execution address). If true, transfers control to the loaded program beginning at the location specified in the C register. If the C register is negative, continues from step 8 below.

5. Places the highest previous location counter value in the X register as the new relocation bias for the next load module.

6. Restores the C register from the contents of the A register.

7. Halts the processor at a prespecified location address (\$EBB) to allow the user the opportunity to ready the next load module in the reader and the option of displaying or altering the X and C register contents. Processing resumes with Step 4 above.
8. Tests for a non-zero value in the X register. If the X register is zero, continues from Step 5 above. If the X register is not zero, the Loader proceeds to process the next load module.

Directive Codes

The Loader analyzes each of the directives contained in the Control Word by loading the Control Word into the 4-byte accumulator and examining the highest order or sign bit. If the accumulator contents are positive, the Loader performs the functions described below for the binary code 0.

If the accumulator is negative, the Loader rotates the 4-byte Control Word one binary position and again tests the highest order or sign bit. If the accumulator is positive, the Loader performs the functions described below for the binary code 10.

If the accumulator is still negative, the Loader rotates the 4-byte Control Word two binary positions, thus separating the two lowest order bits of the four-bit directive code into the sign bit and the even/odd bit of the 4-byte accumulator. The Loader interprets the accumulator status as follows and performs the functions called for by the directive code indicated:

<u>Accumulator Status</u>	<u>Directive Code</u>
even, positive	1100
even, negative	1101
odd, positive	1110
odd, negative	1111

Binary Directive Code: 0

Read and store single-byte data quantity into the next memory location as specified by the Location Counter. Advance the Location Counter by one.

Binary Directive Code: 10

Read two-byte memory reference instruction. Analyze the M-field of the 2-byte instruction to determine if the addressing mode is indirect-through-current-sector or direct, and proceed as follows:

For indirect-through-current-sector addressing mode, compute the effective address of the indirect address operand by offsetting the least significant ten bits (address part) of the two-byte instruction.

If the computed address is inside the current 1024-byte sector, replace only the ten-bit address part of the two-byte instruction with the least significant ten bits of the computed address and transfer the two-byte instruction to memory at the location specified by the Location Counter.

If the computed address is outside of the current 1024-byte sector, store the computed address (with sign bit modified to indicate double indirect addressing) in the next entry of the table of addresses

for indirectly referenced operands. Insert a pointer to the respective table entry into the ten-bit address part of the instruction. Modify the M-field of the instruction to indicate indirect-through-zero-sector addressing and transfer the two-byte instruction to memory at the location specified by the Location Counter.

For direct-through-lower-4-sectors addressing mode, offset the 12-bit address part of the two-byte instruction by the relocation bias.

If the resultant address is inside the lower four sectors of memory, transfer the two-byte instruction to memory at the location specified by the Location Counter.

If the resultant address is not within the lower four sectors of memory, store the resultant address in the next entry of the table of addresses for indirectly referenced operands. Insert pointer to the respective table entry into the ten-bit address part of the instruction. Modify the M-field of the instruction to indicate indirect-through-zero-sector addressing and transfer the two-byte instruction to memory at the location specified by the Location Counter.

Binary Directive Code: 1100

Input two-byte address constant and offset by relocation bias. Transfer the resultant two-byte address constant to memory at the location specified by the Location Counter.

Binary Directive Code: 1101

Input two-byte ASCII-coded ENTRY symbol. Search the Symbol Table using

the ENTRY symbol as the search argument.

If the symbol is found in the Symbol Table and is not flagged as being an undefined symbol, the loader types the following message on the teletype:

```
M      NN      XXXX
```

where:

M is the flag for multiply-defined symbols; NN is the symbol and XXXX is the four-digit hexadecimal location value assigned at the time of the first occurrence of the symbol. The earlier definition is not affected and the present reoccurrence of the symbol is otherwise ignored.

If the symbol is found in the table and is flagged as being undefined, the Loader defines the symbol in the table by assigning to it the value in the Location Counter and resets the undefined flag associated with the symbol. The Loader resolves all previous references to the symbol by inserting the symbol value at each point of reference. These points of reference are links along a chain of memory addresses created by the Loader to tie together all external references associated with the symbol (see functions for Code 1111). The first address or link of the chain is obtained directly from the Symbol Table. Each succeeding link references the next link. The last referenced location or link on the chain will contain all one's bits (-1).

If the symbol is not found in the table, the Loader creates an entry for the

relocating and linking loader

symbol in the Symbol Table and assigns to it the value in the Location Counter.

Binary Directive Code: 1110

Read two-byte memory reference instruction whose twelve-bit address part contains a direct reference to an indirect address operand. Since the indirect address operand referenced by the instruction did not lie inside the current sector at assembly time, the twelve-bit address part of the instruction points directly to the unrelocated indirect address operand (also referred to as an indirect pointer). The Loader offsets the twelve-bit address part of the instruction by the relocation bias to obtain the address of the indirect address operand after the latter is relocated.

If the resultant address references a memory location which is outside of the current 1024-byte sector, the loader stores the resultant address (with sign bit modified to indicate double-indirect addressing) in the next entry of the table of addresses for indirectly referenced operands; inserts a pointer to the respective table entry into the ten-bit address part of the instruction, and modifies the M-field of the instruction to indicate indirect-through-zero-sector addressing. The Loader then transfers the two-byte instruction to memory at the location specified by the Location Counter.

If the resultant address references a memory location which is inside the current sector, the loader retains the least significant ten bits of the resultant address in the ten-bit address part of the instruction, sets the mode bits of the M-field to indicate indirect-through-current-sector addressing, and transfers

the two-byte instruction to memory at the location specified by the Location Counter.

Binary Directive Code: 1111

Input 2-byte ASCII-coded EXTERNAL symbol. Search the Symbol Table using the EXTERNAL symbol as the argument.

If a match is found and the symbol is flagged as being undefined, the chain linking the previous references to the symbol is traced until the end of the chain is located. The value in the Location Counter replaces the one's bits associated with the end of the chain. The all one's bits (−1) replaces the contents of the location referenced by the Location Counter to identify the new end of the chain.

If a match is found and the table symbol is defined, the symbol value is transferred to memory at the location specified by the Location Counter.

If no match is found in the table, the Loader creates an entry for the symbol in the Symbol Table and flags the symbol as being undefined. In addition, the Loader generates the initial link of a chain of memory locations associated with the symbol by retaining, with the symbol in the Symbol Table, the value of the Location Counter as the first location to reference the symbol. All one's bits (−1) are inserted in memory at that location to indicate the end of the chain.

Loader As a Callable Subroutine

The Loader, in relocatable object format, may be used as a callable subroutine. The

following differences apply in the case of the relocatable version.

1. The use of SENSE switches as described in the preceding paragraphs does not apply.
2. The relocatable version of the Loader examines the contents of the byte in the calling program which follows the branch and mark (BRM) instruction linking the calling program with the Loader to determine whether the Symbol Table is to be listed prior to returning to the calling program. If the contents are non-zero, the Loader returns to the location immediately following that byte in the calling program; if the contents are zero, the Loader lists the Symbol Table prior to returning to the calling program.
3. The relocatable version of the loader ignores the last two frames of the object-tape modules which it loads. These frames normally contain an execution address which is overridden by the return address of the calling program.
4. The relocatable version of the Loader does not contain any halts except for the one that occurs when the program being loaded overlaps the Symbol Table. The calling program must include halts to allow the user the opportunity to ready his object tape in the reader.

The following coding sample is an example of a calling sequence to the relocatable version of the Loader:

SP2		
LOD	QR	
TCQ		SET Q REG FOR HSPT INPUT
LOD	RB	
TCX		PUT RELOC BIAS IN X REG
LODI	IP	
SAZN		TEST IF TABLE ADDRESS ALREADY EXISTS*
LOD	IA	PROVIDE INITIAL ADDRESS FOR TABLE OF INDIRECT ADDRESS POINTERS
HLT		READY TAPE IN READER
BRM	L	TRANSFER TO THE LOADER
DAI	0	FLAG TO LIST SYMBOLS (LOADER RETURNS CONTROL TO NEXT BYTE IN SEQUENCE)
EXT	IH	EXTERN LINK FOR HSPT INPUT RTN

The following are the EXTERNAL symbol names which must be used by a calling program to link to the relocating version of the Loader. The symbols may not be used

*Non-zero indicates that the table for indirect pointers already exists. This test insures that the table address will not be reset.

relocating and linking loader

as ENTRY symbols within the calling program:

<u>Symbol</u>	<u>Definition</u>
L	Loader ENTRY point
IT	Teletype input routine ENTRY point

<u>Symbol</u>	<u>Definition</u>
IH	High speed paper tape input routine ENTRY point
IP	Address of next ENTRY in table of Indirect Pointers

SECTION 19 – MATHEMATICAL SUBROUTINES

Features:

- Designed as a system to satisfy many varied mathematical requirements.
- Designed to occupy a minimum amount of core.
- Calling sequences formatted so that subroutines may be used as an extended instruction set.
- Subroutines function in only one environment.
- Subroutines check for error conditions and set special returns to notify caller of errors, or allow continuation without special processing.
- Designed to enable sharing of storage area to conserve core when necessary.
- Subroutines individually coded to comply to a general common format.
- Tables and matrices provided to aid in timing estimates and size estimates for a user's application program.
- Suggested calling sequences and error recovery sequences provided to aid in using the routines.

Mathematical software subroutines for the Varian 520/i are grouped into three major

subsections: fixed point arithmetic, floating point arithmetic, and floating point functions.

The fixed point subroutines are multiply, divide, decimal to binary conversion, and binary to decimal conversion.

The floating point arithmetic subroutines include add, subtract, multiply, divide, integer to floating point conversion, and floating point to integer conversion. There are also several floating point utility subroutines included in this section.

The floating point function subroutines are integer part, fractional part, square root, exponential function, logarithmic function, exponentiation, decimal to binary floating point conversion, binary to decimal floating point conversion, sine, cosine, and arctangent.

Figure 19-1 lists these subroutines and their symbols in alphabetical order. The page numbers refer to the detailed description of individual programs.

The subroutines are all designed to be called by other programs and are intended to fulfill the mathematical requirements of a large class of computer users. The fixed point section is intended for those users who require a minimum, fast arithmetic capability. The fixed point subroutines together with the SUM and SUB machine instruc-

Symbol	Name	Page
AT	Arctangent – Floating Point	19-43
BD	Binary to Decimal Conversion	19-12
CO	Cosine Function – Floating Point	19-42
DB	Decimal to Binary Conversion	19-11
DI	Divide – Fixed Point	19-10
EX	Exponential Function – Floating Point	19-27
FA	Add – Floating Point	19-18
FC	Construct – Floating Point	19-14
FD	Divide – Floating Point	19-22
FE	Exponentiation ($A^{**}B$) – Floating Point	19-31
FI	Floating Point to Integer – Binary	19-17
FM	Multiply – Floating Point	19-20
FN	Normalize – Floating Point	19-15
FP	Fractional Part – Floating Point	19-24
FQ	Floating Point to Decimal Conversion	19-37
FS	Subtract - Floating Point	19-19
IF	Integer to Floating Point	19-16
IP	Integer Part	19-23
LN	Natural Log – Floating Point	19-30
MU	Multiply – Fixed Point	19-8
QF	Decimal to Floating Point Conversion	19-34
RX	Load Index Register – Immediate	19-13
SE	Separate Exponent – Floating Point	19-14
SI	Sine Function – Floating Point	19-39
SQ	Square Root – Floating Point	19-25

Figure 19-1. Index of Mathematical Subroutine Descriptions

tions allow the user to perform integer arithmetic operations and conversions.

The floating point arithmetic is intended for those users who require more accuracy, more flexibility, and greater number ranges. The floating point functions are also intended for those users who, in addition to arithmetic operations, need more powerful mathematical tools.

The fixed point multiply and divide subroutines have been designed to have exact accuracy and fast execution. The floating point subroutines have been designed to maintain accuracy to the least significant bit of the mantissa, and to require a minimum amount of programming preparation. The floating point design also optimizes speed whenever possible without incurring a corresponding size penalty. The results of these design criteria are very tightly coded subroutines which combine to form an integrated system of math routines.

The Varian 520/i mathematical software subroutines have been designed to work together as a system. Where possible, each subroutine calling sequence and exit format has been made compatible with the other subroutine calling sequences and exit formats. This allows a user to execute a string of Branch and Mark instructions (BRM) to the desired math routines to evaluate an expression without a series of formatting instructions between each call. In this way, the math subroutines act as an extended instruction set.

The math subroutines are programmed to utilize only one environment. This allows the user to do simultaneous computing and input-output processing utilizing the dual environment capability of the Varian 520/i.

The number formats and number ranges of the math subroutines have been chosen to provide the most efficient utilization of the computer features and a wide area of application. Binary integers may be up to 32 bits long. Floating point numbers have an exponent range of 2^{-64} to 2^{63} (approximately 10^{-19} to 10^{18}). The floating point mantissa is 24 bits long, which provides approximately 7 significant decimal digits.

The routines check for error conditions such as underflow, overflow, incorrect sign, and other conditions which are unacceptable for specific routines. If error conditions are found, the routine returns with an indication of the error found. In the case of overflow or underflow, the largest or smallest number is returned to allow continuation of calculations when the user's accuracy requirements will allow this.

General Description

All of the math subroutines are designed in a general format to facilitate using the routines as a system. Arguments and parameters are passed to and from the subroutines in the A, C, and X registers. A minimum amount of this information is altered by the operation of the subroutine.

If a floating point subroutine has only one argument, it is passed in the 4-byte accumulator. If the subroutine has two arguments, one will be in the 4-byte accumulator and the other in memory at an address specified in the X register. Results will be returned in the 4-byte accumulator. All subroutines operate in one environment and use or alter only that environment's registers.

Label Format

The math subroutines are written using a label format which allows all of the routines to be assembled together, and at the same time allows the user to differentiate between possible math subroutine labels and all other labels.

Each math subroutine has a unique symbolic name (see Figure 19-1). This name is used as the operand of the BRM instruction which calls the subroutine. Each subroutine has associated with it a unique letter. This letter is used as the first character of all labels used in the routine. The second character of any label in a math routine is a digit. Therefore, there is no possibility of label conflict as long as the user does not use the symbolic name of any math routine assembled with the main program, or use a letter followed by a digit. More than 675 symbol combinations are available that meet these requirements.

In each math subroutine, the digit part of the label generally implies the use of the label. The digit zero identifies the starting address of the temporary storage block when temporary storage is required. Nine identifies a two byte area used as temporary storage for the X register. These two bytes are imbedded in the routine itself but may still be used as temporary storage when the routine is not active. Digits one and greater are used for branch points within the math routine. Digits eight and less are used for additional imbedded temporary storage or for data constants.

Number Formats

Fixed point numbers are 16-bit and 32-bit signed binary integers with the following characteristics:

- A negative binary integer is represented in 2's complement form.
- 16-bit integer range is less than 1^{15} (32,768 decimal) and greater than or equal to -2^{15} .
- 32-bit integer range is less than 2^{31} (2,147,483,648 decimal) and greater than or equal to -2^{31} .
- The decimal integer is a Header Cell* followed by one to ten ASCII coded digits (one byte each).

Floating point numbers are 4-byte (32 bit) numbers with the following characteristics:

31 30 24 23 0

Exponent	Mantissa
----------	----------

Mantissa Sign - 0 = positive, 1 = negative

- The mantissa is a 24-bit positive binary fraction whose absolute range is from zero to 8,388,608 decimal, or seven significant decimal digits. It is assumed that the mantissa is normalized. All routines normalize the output number.

*A Header Cell is a one-byte binary integer indicating the number of digits in the decimal number. The Header Cell contents are 2's complemented if the sign of the decimal number is negative.

- The exponent is a seven bit signed binary number whose range is from -64 to +63. A negative exponent is represented in 2's complement form.
- The range of numbers which can be represented in the floating point format is from 5.421011×10^{-20} to 9.22337×10^{18} decimal.
- The floating point decimal number is a Header Cell followed by one to ten ASCII coded fraction digits, followed by a Header Cell, followed by one or two ASCII coded decimal digits representing the exponent of ten.

Operating Requirements

The math subroutines will operate in the minimum configuration of a Varian 520/i computer with 4K of memory and a model 33TC teletype. There are no limitations on using the subroutines with any other configuration of the Varian 520/i.

The math subroutines are designed to operate in one environment only of the 520/i without altering the conditions of the other environment. The one environment may be either the interruptable or the noninterruptable environment. Sense switches are not used.

Several of the math subroutines operate independently, but most of them call other math subroutines to perform their function. A matrix of which are called by each subroutine, the size of each subroutine, and the total size of each routine along with the routines it calls, are given in Figure 19-2.

Figure 19-3 presents the minimum, maximum and average time of execution of each routine including the time consumed by each subroutine called by a particular math subroutine.

Assembly of Math Subroutines with Symbolic Assembler, Version A

When assembling the math routines and a user's program separately, the main concern is linkage between the routines and, if space is limited, sharing of temporary storage.

Using the Symbolic Assembler, Version A, linkage between the routines can be accomplished by using EQU statements in the main program. These EQU statements define the symbolic name of the routine used and give the location at which it is to be found. Sharing of temporary storage is handled in a similar manner. For each math routine which is to share temporary storage, the user replaces the statement in the math routine which has a label consisting of a letter followed by the digit 0. The associated data statement is replaced with an EQU statement having the same symbol as its label and the location of the common temporary storage block as its operand. Care must be exercised when performing this operation because of addressing problems and nested calls to other math subroutines. In general, two math subroutines may share temporary storage only if the subroutines do not call each other.

The math routines are supplied to the user in the form of source punched paper tape.

Name	Routine	Size* (bytes)	Temp. Storage	Name	RX	SE	FN	IF	FI	FA	FS	FM	FD	FP-IP	LN	EX	SO	BD	Total Size*
RX	Load X Immediate	22			22	31	31	76	61	77	152	138	160	112	117	206	183	164	2464
SE	Separate Exponent	R			R														31
FC	Construct Floating Point				R														31
FN	Normalize																		76
IF	Integer to Float																		61
FI	Float to Integer																		77
FA	Add				X	X	X	X	X	X	X	X	X	X	X	X	X	X	281
FS	Subtract				I	I	I	I	I	I	I	I	I	I	I	I	I	I	317
FM	Multiply																		138
FD	Divide																		160
FP-IP	Fractional Part - Integer Part				X	X	X	X	X	X	X	X	X	X	X	X	X	X	272
SQ	Square Root				X	X	I	I	I	I	I	I	I	I	I	I	I	I	558
EX	Exponential e ^X				X	X	X	X	X	X	X	X	X	X	X	X	X	X	1045
LN	Natural Log				X	X	I	X	I	X	I	X	I	X	I	X	I	X	865
FE	Exponentiation A**B				I	I	I	I	I	I	I	I	I	I	I	I	I	I	1310
QF	Decimal to Float				X	X	X	I	X	I	X	I	X	I	X	I	X	I	1379
FQ	Float to Decimal				X	X	X	I	X	I	X	I	X	I	X	I	X	I	1405
SI	Sine				X	I	I	I	I	I	I	I	I	I	I	I	I	I	974
CO	Cosine				I	I	I	I	I	I	I	I	I	I	I	I	I	I	999
AT	Arctangent				I	I	I	I	I	I	I	I	I	I	I	I	I	I	824

*Size includes temporary storage size

Figure 19-2. Matrix of 520/i Floating Point Math Subroutine Size and Interaction

Routine Symbol and Name		Storage – bytes		Time-Cycles		Average Time Milliseconds	Routine Label
		Routine	Temp	Minimum	Maximum		
FIXED POINT							
MU	Multiply – 2 byte	107	4	101	154	.19	U
DI	Divide – 2 byte	133	10	261	274	.40	D
DB	Decimal to Binary	98	8	106	880	.70	K
BD	Binary to Decimal	154	10	154	2084	2.00	B
FLOATING POINT							
RX	Load X Immediate	20	2	35	35	.05	X
SE	Separate Exponent	28	3	34	42	.05	F
FC	Construct Floating Point	28	3	36	42	.06	C
FN	Normalize	70	6	82	261	.26	N
IF	Integer to Float	54	7	74	319	.35	J
FI	Float to Integer	70	7	71	346	.30	I
FA	Add	144	8	336	634	.74	A
FS	Subtract	32	4	388	687	.82	S
FM	Multiply	128	10	909	996	1.42	M
FD	Divide	146	14	1125	1236	1.76	V
FP	Fractional Part			127	648	.60	P
IP	Integer Part	110	2	124	704	.60	P
SQ	Square Root	105	12	5090	5455	7.83	Q
EX	Exponential e ^X	198	10	9440	9950	14.54	E
LN	Natural Log.	173	10	----	----	15.82	L
FE	Exponentiation A**B	23	0	----	----	31.81	Z
QF	Decimal to Float	163	6	----	----	18.15	W
FQ	Float to Decimal	138	8	----	----	22.80	R
SI	Sine	206	10	13250	16100	19.95	O
CO	Cosine	27	0	13700	16500	20.50	O
AT	Arctangent	203	6	----	----	27.00	T

NOTES

1. One cycle equals 1.5 microseconds
2. Times include the time of all other routines called during execution.

Figure 19-3. Summary of 520/i Math Subroutine Size and Timing

mathematical subroutines

Each routine begins with a comment which identifies it and ends with a MOR statement, followed by an END statement, followed by 10 inches of leader. Subroutines which the user requires may be selected from the math tape and assembled with the user's program. Each subroutine tape stops after the MOR statement. If additional subroutines are to be used, the tape is positioned to another routine. If no additional routines are required, the Assembler is restarted, and the END statement is read.

Assembly of Math Subroutines with Symbolic Assembler, Version B

A relocatable object tape of all of the math subroutines, each assembled separately at zero with the Symbolic Assembler, Version B (Relocatable Object Code), is supplied to the user. The subroutines are assembled in the order listed in Figure 19-2, and are each separated by 10 inches of blank leader (exceptions to this are that IP and FP are assembled together and SI and CO are assembled together).

A particular math subroutine is used by loading it, plus any subroutines that it calls, and by calling the subroutine from the user's program with a BRM instruction. All necessary EXT and ENT pseudo operand codes are used. The subroutines and the user's program or programs are then loaded using the Varian 520/i Relocating and Linking Loader as described in Section 18.

Subroutine Descriptions

The subroutine descriptions on the succeeding pages have the following format:

1. Subroutine general description.
2. Register contents required on entry and register contents on exit.
3. Error conditions checked, and register contents reflecting error conditions.
4. Coding sequence which supplies the required entry conditions and tests for the indicated error conditions. These coding sequences, in most cases, are not used as stated because many entry conditions are already satisfied by a user's program or by a previously called math subroutine's exit condition.
5. Memory requirements for the routine and time of execution of the subroutine in cycles, including the time required for all subroutines called during the execution of the subroutine.
6. Detailed description of the subroutine, indicating the method used, the accuracy of the results, formats required and used, unusual conditions, and any cautions to the user.
7. Subroutines required for execution, called by the subroutine.

MU Fixed Point Multiply

1. Fixed point multiply forms the arithmetic product of two signed 16 bit binary integers. The product returned is a 32 bit binary integer whose sign is algebraically correct.

- On entry, the A register contains the multiplicand and the C register contains the multiplier.

On exit, the 4-byte accumulator contains the product, the X register is unchanged, overflow is reset and precision is set to 4.

- Fixed point multiply has no error conditions or error exits.

- Coding sequence:

```
SP2
LOD    MULTIPLICAND
TCA                      TRANSFER
                        TO A
                        REGISTER
LOD    MULTIPLIER
BRM    MU
```

- MU is 107 bytes long plus 4 bytes of temporary storage.

The time for execution of MU is from 101 to 154 cycles. The minimum time occurs if the numbers are positive and the multiplier is zero. The maximum time is required if the numbers are negative and the multiplier is all ones.

- Fixed point multiply is accomplished by computing the product sign, multiplying the two positive numbers to obtain the 32-bit product, and returning the properly signed product to the caller. The multiplication itself is done by repetitive shifting and adding the multiplicand into the accumulating product each time the multiplier bit being tested is one. For example, multiplying 110 by 101 is equivalent to adding 110, shifting

right once and not adding (bit is zero) and shifting right again and adding 110.

```
110 multiplicand
101 multiplier
-----
110
000
110
-----
011110 product
```

In the case of the MU subroutine, the multiplication consists of 16 one bit multiplies where a one bit multiply is a "conditional addition" and a right shift of the partial product. A conditional addition of the multiplier to the most significant 16 bits of the partial product if and only if the least significant bit of the multiplicand is one. The 4-precision right shift one of the multiplicand product following the conditional addition serves two purposes. First, it positions the next bit of the multiplicand into the least significant position to be tested for the next iteration. Second, it shifts the product right, which is equivalent to shifting the multiplicand left for the next conditional addition.

The most significant half of the 32 bit partial product is cleared to zero before the first one bit multiply. Each successive one bit multiply uses the new partial product produced by the previous multiply. After the last one bit multiply the partial product is the desired product in positive form. It is necessary to do 16 one-bit multiplies instead of 15 to accommodate the largest negative number in 16 bits (\$8000). Normally, the most significant bit, the sign bit, of the multiplicand will be zero since the multiplicand is positive. However, the largest negative number is left unchanged after complementation. Therefore, the addition

mathematical subroutines

in the 16th one-bit multiply is only performed when the multiplicand is the largest negative number.

After completion of the multiplication, the positive product is given the previously determined product sign (complemented if negative) and the X register is restored. The resultant 32 bit product is correct for any multiplier-multiplicand combination. The coding of the multiplication portion of the MU subroutine is repeated 16 times rather than coded once and looped through 16 times. This is done to enable a fast execution of the multiplication.

DI — Fixed Point Divide

1. DI forms the quotient of a signed 32-bit binary integer dividend divided by a signed 16-bit binary integer divisor. The quotient returned is a signed 16-bit binary integer, and a 16-bit positive remainder is also returned. The non-restoring method is used.

2. On entry, the four-byte accumulator contains the dividend, and the X register contains the divisor.

On exit, the C register contains the quotient, the A register contains the remainder, the X register is unchanged, overflow is reset, and precision 2 is set.

3. The 16-bit divisor must be twice as large as the most significant 16 bits of the dividend (greater than twice as large if the quotient is positive). If this condition is not met, overflow is set and an unspecified quantity is returned in the 4-byte accumulator. The X register remains unchanged and precision 4 is set.

4. Coding sequence:

SP2	
LOD	DIVISOR
TCX	TRANSFER DIVISOR TO X REGISTER
SP4	
LOD	DIVIDEND
BRM	DI
SOVN	TEST FOR DIVIDE ERROR
BRU	ERROR EXIT

5. DI is 123 bytes long plus 10 bytes of temporary storage.

The time for execution of DI is from 260 to 274 cycles depending upon the signs of the numbers involved.

6. The software divide is accomplished by computing the product sign, taking the positive dividend and divisor, testing for divide error, dividing the numbers, and returning the positive remainder and properly signed quotient. After the sign is computed and stored and the numbers are made positive, the divisor is increased to 4 bytes by adding two bytes of zeros in the right half of the 4-byte accumulator. This 4-byte divisor is subtracted from the dividend and the result is tested. If the result is positive, the divide error exit is taken which returns with overflow set. If the result is zero, the quotient sign is tested. If the quotient sign is positive, the error exit is taken. If the sign is negative, the divide continues.

The actual division is done by executing the following sequence 15 times:

- a. The divisor is subtracted from the dividend.
- b. The sign of the result is tested.
 1. If positive, one is put in the quotient.
 2. If negative, a zero is put in the quotient.
- c. The remaining dividend is shifted left once.
- d. The divisor is either added to or subtracted from the dividend depending on the previous result (last quotient bit).
 1. If the previous quotient bit is zero, the divisor is added.
 2. If the previous quotient bit is one, the divisor is subtracted.
- e. The process is repeated until the divide is completed (16 times) by looping to b.

The particular implementation of this method in the DI subroutine is as follows:

- a. The dividend is placed in the 4-byte accumulator and the divisor is in memory in 4 bytes, the lower two being zeros.
- b. The divisor is subtracted from the dividend.
- c. The result is rotated left once placing the sign bit in the least significant position of the accumulator. This bit is now the

least significant bit of the quotient (complemented) which will accumulate in the C register. The accumulator is now in position for the next iteration.

- d. The least significant bit is tested (odd or even).
 1. If it is zero, the divisor is subtracted from the accumulator.
 2. If it is one, the divisor is added to the accumulator.
- e. The process is repeated 15 times by looping back to c.

The result of the above process is the complement of the quotient in the C register, and the remainder in the A register. If the remainder is negative, the divisor is added to the remainder once to restore it to its proper value. The quotient is complemented and signed properly and the division is complete. The sign of the remainder is left positive.

DB – Decimal to Binary Conversion

1. The DB subroutine converts a decimal integer into a four byte binary number. The number to be converted may be from one to ten digits long, in ASCII code or 8-bit BCD code.
2. On entry, the X register must contain the address of the Header Cell of the number to be converted.

On exit, X is unchanged, the 4-byte accumulator contains the number converted, precision is 4, and overflow is reset.

mathematical subroutines

3. If the Header Cell value is greater than 10, or the input number is too large, overflow is set and zero is returned in the accumulator. Results will be meaningless if the digit field does not contain digits.

4. Possible coding sequence:

SP2

LOD ADDRESS OF HEADER
 CELL

TCX TRANSFER ADDRESS
 TO INDEX REGISTER

BRM DB

SOVN ERROR TEST

BRU ERROR EXIT

5. DB is 98 bytes long plus 8 bytes of temporary storage. The time for execution of DB is 25 to 34 cycles plus $81 \cdot N$ cycles where N is the number of digits to be converted.

6. The format of data for the DB routine is a Header Cell followed by one to ten decimal digits in memory. The Header Cell contains a number indicating how many digits are to be converted. The number is signed (2's complement form) to indicate the sign of the number to be converted. The decimal digits are intended to be ASCII coded, but may also be binary coded decimal form, one digit per byte, right justified. The conversion is done by repetitive multiplication by 10 and adding of the next digit. The multiplication by 10 is done by shifting left (equivalent to multiplying by 2) and adding. For example, multiplying X by 10 is done by a shift ($2X$) and then two more shifts ($8X$) and then

adding the two quantities together ($2X + 8X = 10X$). This process is repeated until the specified number of digits have been converted. The accuracy of the conversion is exact, provided the correct format is supplied.

BD – Binary to Decimal Conversion

1. The BD subroutine converts a four-byte binary number in the accumulator into ASCII coded decimal digits in an area specified in memory.
2. On entry, the 4-byte accumulator contains the binary number to be converted, and the X register contains the address of the Header Cell of the area in which the results are to be stored.

On exit, the X register is unchanged, the C register has the same content as the X register, the A register is unspecified, precision is 2, and overflow is reset.

3. An error is indicated by setting overflow if the number is longer than the length specified in the Header Cell or if the number is -2^{31} (the largest negative number).
4. Coding sequence:

SP2

LOD ADDRESS OF STORAGE
 LOCATION

TCX TRANSFER ADDRESS
 TO INDEX REGISTER

LOD SIZE OF NUMBER
 (NUMBER OF DIGITS)

STOX O STORE SIZE IN HEADER
 CELL

SP4	
LOD	BINARY NUMBER TO BE CONVERTED
BRM	BD
SOVN	ERROR TEST
BRU	ERROR EXIT

5. BD is 154 bytes long plus 10 bytes of temporary storage.

The time for execution of BD is from 61 to 74 cycles plus

$$\sum_{i=1}^N (78 + 15d_i)$$

cycles where N is the number of digits requested to be converted and d_i is the i th digit.

6. The input to the BD routine is a 4-byte signed binary integer in the accumulator and a memory address in the X register. The contents of memory at the address in the X register (the one-byte Header Cell) may be a signed number from one to ten indicating how many decimal digits are to be contained in the converted decimal number.

If the content of the Header Cell is other than a signed value between one and ten, BD will place the value ten in the cell and convert 10 digits. If the value in the Header Cell is smaller than the number of decimal digits in the converted number, an error exit occurs. The sign of the binary number to be converted is tested and if the number is negative, the Header Cell is complemented to indicate the sign to the user.

The conversion is started by subtracting the power of ten which is one less than the number in the Header Cell. For example, if 7 digits are requested, one million (10^6) will be subtracted from the number. The quantity is subtracted until the number goes negative, then the number of times it was subtracted less one is the most significant decimal digit.

This process is repeated for the next smaller power of ten to determine the next digit, and continued until the one's digit is evaluated and the value of the binary number is reduced to zero. After each digit is evaluated, it is converted to ASCII code and stored in the specified area of memory. The accuracy of the conversion is exact.

RX — Load X Immediate

1. RX is a utility routine used by other floating point math subroutines. It loads the X register with the 2 bytes following the call instruction without altering any other conditions.
2. On entry, no conditions are specified, and on exit, all conditions remain unchanged except the X register, which contains the 2 bytes following the call instruction, and the precision which is set to four.
3. There are no error conditions checked by RX.
4. Coding sequence:

BRM	RX
DA2	DATA TO BE LOADED INTO X REGISTER

mathematical subroutines

5. RX requires 20 bytes of memory plus 2 additional bytes for temporary storage. The execution of RX takes 35 cycles to complete.
6. Since the X register may not be loaded directly from memory by a machine instruction, the RX routine is provided to accomplish this. Without RX, it is necessary to first save the contents of the C register, load the desired data into C, transfer C to the X register, and restore the original contents of the C register to continue operation. In addition, it is sometimes necessary to change precision. The RX routine does exactly what is described above and returns to the user's program following the two bytes of data which were loaded into the X register. Using RX, the user can have the index register loaded with 5 bytes of coding as opposed to as many as 10 bytes of coding without RX. Repeated use of this routine throughout the floating point math routines saves a large amount of core.

SE — Separate Exponent and Mantissa

1. SE separates the exponent from the mantissa of a floating point number, returns the signed mantissa in the 4-byte accumulator, and returns the signed exponent in the 2-byte X register.
2. On entry, the 4-byte accumulator contains a floating point number.

On exit, the 4-byte accumulator contains the mantissa, the X register contains the exponent, overflow is reset, and precision is set to 4.
3. No error conditions are detected in SE.

4. Coding sequence:

SP4

LOD

FLOATING POINT
NUMBER

BRM SE

5. SE requires 28 bytes of memory plus 3 bytes of temporary storage. SE will function in from 34 to 42 cycles depending upon the signs of the exponent and mantissa.
6. The SE routine separates and saves the mantissa from the floating point number, tests and saves the sign of the mantissa in the overflow register, and proceeds to format the exponent. The exponent is formatted by right justifying it in the accumulator, and repeating the sign bit in the upper 9 bits of the 16-bit C register. The exponent is then transferred to the X register. The mantissa is reloaded into the 4 byte accumulator with the upper byte cleared to zero. If the sign of the mantissa (as saved in the overflow register) is negative, the 4-byte accumulator is complemented. Overflow is reset, and the control returns to the user with the mantissa in the 4 byte accumulator, and the exponent in the X register, both with the sign extended to the left.

FC — Construct Floating Point Number

1. FC constructs a floating point number given the mantissa in the 4-byte accumulator and the exponent in the X register. This routine is the inverse of the SE routine.

2. On entry, the 4-byte accumulator contains the signed mantissa and the X register contains the signed exponent,

On exit, the 4-byte accumulator contains the floating point number, the X register is unchanged, overflow is reset, and the precision is set to four.

3. No error conditions are checked or indicated,
4. Coding sequence:

SP2

LOD EXPONENT

TCX TRANSFER EX-
 PONENT TO X
 REGISTER

SP4

LOD MANTISSA

BRM FC

5. The FC routine requires 28 bytes of memory plus 3 bytes of temporary storage. The time of execution is from 36 to 42 cycles depending upon the signs of the exponent and mantissa.
6. The FC routine tests the sign of the mantissa, saves it in the overflow register and then makes the mantissa positive if it was negative. The 24 significant bits of the accumulator are saved for the mantissa, and the exponent is transferred to the 2 byte accumulator. The upper 9 bits of the 16 bit exponent are shifted out of the accumulator, and the sign bit of the mantissa, as indicated in the overflow register, is placed in the most significant place of the accumulator. The exponent-sign information is placed to

the left in the 4-byte accumulator, and the 3 bytes of the mantissa are loaded into the lower portion of the accumulator, completing the construction. The user is cautioned that the 4-byte mantissa supplied to FC must have only 24 significant bits right justified, and the exponent must have only 7 significant bits right justified. The signs of these quantities must be extended to the left to complete the respective register contents.

FN – Floating Point Normalize

1. FN normalizes a floating point number by left-shifting the mantissa and decrementing the exponent until the most significant bit of the mantissa is a one.
2. On entry, the 4-byte accumulator contains the floating point number to be normalized.

On exit, the 4-byte accumulator contains the floating point number normalized, the X register contents are unspecified, overflow is reset, and precision 4 is set.

3. If exponent underflow occurs during the normalization process, overflow is set, and all zeros are returned in the 4-byte accumulator.
4. Coding sequence:

SP4

LOD FLOATING POINT
 NUMBER TO BE
 NORMALIZED

BRM FN

SOVN TEST FOR ERROR
 (OPTIONAL)

BRU ER ERROR EXIT

5. FN requires 70 bytes of memory plus 6 bytes of temporary storage. The time for execution of FN is from 82 to 261 cycles depending upon the number of leading zeros in the mantissa.
6. All routines in the floating point math system assume that the floating point numbers supplied are normalized, and exit with the floating point number normalized. The normalization is usually done by calling the FN routine. On entry, FN tests if the number is already normalized. If it is, the number is returned immediately. The mantissa is tested for zero. If the mantissa is zero, the entire 4-byte accumulator is cleared and returned.

The exponent and sign are separated from the mantissa, the mantissa sign is tested, and if negative, the overflow register is set.

The exponent is then shifted left once (which doubles it and removes the sign bit) and then saved. The X register is cleared to count the mantissa shifting. A loop is executed which shifts the mantissa left, decrements the X register, and tests the most significant mantissa bit for a one. The count is shifted left and the mantissa sign bit is placed in the right most bit. The left-shifted exponent is added to the count. If overflow occurs, the total exponent was less than minus 64 (shifted left once = -128) which is an underflow condition and the error exit is taken. Otherwise, the

exponent is rotated right once, which compensates for the left shift and correctly positions the mantissa sign bit. The exponent and sign are combined with the mantissa, and returned.

IF — Fixed Point to Floating Point Conversion

1. IF converts a 4-byte (32 bit) signed binary integer to floating point format.
2. On entry, the 4-byte accumulator contains the binary integer to be converted.

On exit, the 4-byte accumulator contains the floating point number, the X register is unchanged, precision 4 is set, and overflow is reset.

3. No error conditions are detected in IF.
4. Calling sequence:

SP4

LOD BINARY INTEGER TO
 BE CONVERTED

BRM IF

5. IF is 54 bytes long plus 7 bytes of temporary storage. The time for execution of IF is from 74 to 319 cycles depending upon the number of leading zeros in the binary integer and the sign of the integer.
6. The input to IF is a 32-bit signed binary integer. The X register contents on entry are saved so that the X register may be used as a counter. The integer is made positive, if negative, and the sign is saved in the overflow register. The integer is tested for zero. If the number is zero,

the routine returns immediately because the floating point representation of zero is all zeros.

The X register is initialized to zero. The 32-bit accumulator is shifted left, the X register decremented, and the most significant bit of the accumulator tested until it is a one. When the most significant bit is one, the most significant 24 bits of the accumulator are stored in the mantissa position of the floating point number. Thirty-two (decimal) is added to the negative count of shifts to normalize the number. The resultant number is the exponent of the fractional mantissa. The sign of the number is added to the most significant position, the sign-exponent is combined with the mantissa, the X register is restored, and the routine returns.

FI – Floating Point to Fixed Point Conversion

1. FI converts a floating point number into a 32 bit signed binary integer. Any fractional part of the number is lost.
2. On entry, the 4 byte accumulator contains the floating point number to be converted.

On exit, the 4 byte accumulator contains the converted, signed binary integer, the X register is unchanged, overflow is reset, and precision is set to four.
3. Two error conditions are possible. The floating point number may be either too large (greater than $2^{31} - 1$) or too small (fractional) to express as a 32 bit binary integer. If the number is too large,

overflow is set and if the number is positive, $2^{31} - 1$ (\$7FFF) is returned. If the number is negative, -2^{31} (\$8000) is returned. If the number is too small, overflow is not set and zero is returned.

4. Coding sequence:

SP4		
LOD		FLOATING POINT NUMBER
BRM	FI	
SOV		TEST FOR OVER- FLOW
BRU*+7		NO OVERFLOW, SKIP 7 BYTES
SAN		TEST FOR SIGN
BRU	E1	OVERFLOW AND POSITIVE
BRU	E2	OVERFLOW AND NEGATIVE

5. FI is 70 bytes long plus 7 bytes of temporary storage.

The time for execution of FI is from 71 to 346 cycles depending upon the number of shifts required to adjust the mantissa into position.

6. The floating point number input to FI is saved and the sign of the mantissa is saved and stored in the overflow register. The X register is saved. The exponent is tested to determine if the floating point number is in the range which can be converted. If the exponent is negative, the number is a fraction and zero is returned. The decimal constant 32 is subtracted from the exponent and if the result is not negative, the number is too

mathematical subroutines

large. Either the largest (positive) or the smallest (negative) number is returned, with overflow set depending on the mantissa sign.

Now the quantity in the accumulator (exponent minus 32) represents the number of times the mantissa must be right shifted in the 4-byte accumulator to represent the number. The shifting is done in a loop with the count in the X register. The loop is complete when the X register is zero. The resultant binary integer is signed by complementing if overflow is set. The X register is restored. If the number is part integer and part fraction, the fraction part is lost without rounding off. For example, 2.9 floating point would be converted to binary integer 2. The floating point number -2^{31} , the smallest negative fixed point number, is converted correctly, but overflow is set indicating the number is too small.

FA — Floating Point Addition

1. FA adds two floating point numbers by adding the number whose address is in the X register to the number in the accumulator.
2. On entry, the 4-byte accumulator contains the floating point addend, and the X register contains the address of the floating point augend.

On exit, the 4-byte accumulator contains the floating point sum, the X register and the augend in memory are unchanged, the overflow register is reset, and precision four is set.

3. An error is indicated if either exponent underflow or exponent overflow occurs. The error is indicated by returning with overflow set. If underflow occurred, zero is returned in the 4-byte accumulator. If overflow occurred, the largest possible floating point number is returned in the accumulator with the mantissa having the sign of the sum (\$FFFFFFFF or \$BFFFFFFF).

4. Coding sequence:

SP2		
LOD		ADDRESS OF AUGEND
TCX		TRANSFER TO INDEX REGISTER
SP4		
LOD		ADDEND
BRM	FA	
SOV		TEST FOR ERROR
BRU	*+7	NO ERROR
SAZ		TEST FOR OVER OR UNDERFLOW
BRU		OVERFLOW ERROR
BRU		UNDERFLOW ERROR

5. FA is 144 bytes long plus 8 bytes of temporary storage. The time of execution of FA is from 336 cycles to 634 cycles, depending upon the amount of adjusting required to match the exponents.
6. Floating point addition requires that the two numbers to be added must have equal exponents. Since FA must be able to add any two floating point numbers, the exponents must be equalized. To do

this, the smaller exponent must be made equal to the larger, and the mantissa of the smaller exponent must be right shifted the number of times that the exponent is increased (de-normalized).

FA implements the addition by executing the following steps:

- a. Save X register.
- b. Separate the exponent and mantissa of addend (A) and save in memory.
- c. Load augend (B), separate exponent and mantissa and save in memory.
- d. Determine which exponent is larger, save the larger for the sum exponent and determine the difference.
- e. De-normalize the smaller exponent mantissa the amount indicated by the difference in d.
- f. Add the two numbers, test for mantissa overflow. Increment exponent in X, and adjust mantissa if mantissa overflow occurs.
- g. Test for exponent overflow (caused by adjustment in f, if exponent was 63) and take overflow exit if present.
- h. Construct the floating point sum, normalize the sum, restore the X register, and return.

The normalization is done by calling the FN routine which checks for exponent underflow and returns with a cleared accumulator and overflow set when underflow occurs. The exponent overflow error tested in step g is also

indicated by overflow set, and the largest possible floating point number with the sign of the sum is returned. These error exits enable the user to omit error checking and use the approximations supplied by FA when accuracy requirements will allow this.

The user is cautioned that when two numbers which are very close in value and of opposite sign are added, the result has fewer significant figures and may lead to incorrect calculations. With the exception of the case mentioned above, the accuracy is 24 bits in the mantissa.

7. Subroutines called: SE, FN, RX.

FS — Floating Point Subtract

1. FS subtracts a floating point number stored in memory at the address in the index register from the floating point number contained in the 4-byte accumulator.
2. On entry, the 4-byte accumulator contains the floating point minuend, and the X register contains the address of the floating point subtrahend contained in memory.

On exit, the 4-byte accumulator contains the difference, the X register and subtrahend are unchanged, the overflow register is reset, and precision 4 is set.

3. An error is indicated when exponent underflow or overflow occurs by setting the overflow register. Zero is returned in the case of underflow, and the largest possible number with the sign of the difference is returned in the case of overflow.

mathematical subroutines

4. Coding sequence:

SP2

LOD ADDRESS OF SUBTRA-
 HEND

TCX TRANSFER ADDRESS
 TO INDEX

SP4

LOD MINUEND

BRM FS

SOV TEST FOR ERROR

BRU *+7 NO ERROR

SAZ TEST FOR TYPE OF
 ERROR

BRU OVERFLOW ERROR
 EXIT

BRU UNDERFLOW ERROR
 EXIT

5. FS is 32 bytes long plus 4 bytes of temporary storage.

The time of execution of FS is from 388 cycles to 687 cycles depending upon the amount of adjusting required to match exponents and then normalize the answer.

6. Floating-point subtract is implemented by:

- a. Saving the minuend and the X register.
- b. Loading the subtrahend from memory and changing the sign of its mantissa.
- c. Entering the floating-point add routine after adjusting the return address

so that FA will return directly to the user's program.

The FA routine description describes the remaining steps taken to do a floating-point subtract.

The subtrahend is complemented and restored in the FS routine temporary storage for the FA routine. This is done instead of storing back into memory so that the subtrahend remains unchanged allowing the user to use it repeatedly. The caution and accuracy given for FA apply for FS.

7. Subroutine called: FA, which in turn calls SE, FN, and RX.

FM – Floating Point Multiply

1. FM multiplies the floating point number in the accumulator by a floating point number in memory at the address contained in the X register.
2. On entry, the 4-byte accumulator contains the floating point multiplicand and the 4-byte multiplier is contained in memory at the address contained in the X register.

On exit, the 4-byte accumulator contains the floating point product, the X register and the multiplier in memory are unchanged, overflow is reset, and precision 4 is set.

3. An error is indicated when exponent underflow or overflow occurs by setting the overflow register. Zero is returned in the case of underflow, and the largest possible number with the product sign is returned in the case of overflow.

4. Coding sequence:

SP2	
LOD	ADDRESS OF MULTIPLIER
TCX	TRANSFER ADDRESS TO INDEX REGISTER
SP4	
LOD	MULTIPLICAND
BRM	FM
SOV	TEST FOR ERROR (OPTIONAL)
BRU	*+7 NO ERROR, CONTINUE PROGRAM
SAZ	TEST TYPE OF ERROR
BRU	OVERFLOW ERROR EXIT
BRU	UNDERFLOW ERROR EXIT

5. FM requires 128 bytes of memory plus 10 bytes of temporary storage. The time of execution of FM is from 909 to 996 cycles, depending upon signs and the number of one bits in the multiplier.

6. Floating-point multiply is implemented by executing the following sequences:

- Store the mantissa of the multiplicand after shifting it right once to prevent overflow during multiplication.
- Store the mantissa of the multiplier.
- Initialize the product to zero.
- Calculate and save the product sign by executing an exclusive or operation with the multiplier and multiplicand sign bits.

e. Calculate and save the exponent of the product by adding the multiplier and multiplicand exponents.

f. Perform the multiplication by executing the following steps 24 times:

- Test the least significant bit of the multiplier, set overflow flag if one, shift multiplier right for next pass and store it.
- Load the partial product, add the multiplicand to it if the flag is set, shift partial product right once for next pass and store it.
- Load the multiplier, test if it is zero (indicates 24th pass) and loop back to 1. Advance to g when the multiplier is zero.

g. Adjust the product (shift left one) to compensate for the adjustment in step a, and test for zero product.

h. Test to see if the product is normalized. If it is not, shift it left one to normalize it and set the overflow flag to indicate the operation.

i. Load the product exponent, test overflow indicating the mantissa shift. If it is set, decrement the exponent.

j. Test if exponent underflow or overflow occurred by testing if the exponent is in the range: $64 > \text{exp.} \geq -64$. Take the error exit if the exponent is out of range.

k. Construct the floating point product by assembling the product sign, exponent, and mantissa into the floating point format and return to called.

The error exit, taken when exponent overflow or underflow is detected, sets the overflow register. If underflow occurred, zero is returned in the accumulator. If overflow occurred, the largest possible number with the sign of the product is returned. In many cases, the error return format will allow the user to continue his program using the approximation returned instead of taking special error exits. The accuracy of FM is 23 bits in the mantissa.

FD — Floating Point Divide

1. FD divides the floating-point dividend in the accumulator by the floating point divisor in memory at the address contained in the X register.
2. On entry, the 4-byte accumulator contains the floating-point dividend, and the 4-byte floating-point divisor is contained in memory at the address contained in the X register.

On exit, the 4-byte accumulator contains the floating-point quotient, the X register and the divisor in memory are unchanged, overflow is reset, and precision 4 is set.

3. An error is indicated by a return with the overflow register set when exponent overflow or underflow occurs, and when the divisor is zero. Zero quotient is returned in the case of overflow. The largest possible number with the quotient sign is returned in the case of overflow. The largest possible positive number is returned in the case of zero divisor.

4. Coding sequence:

- | | | |
|-----|-----|------------------------------------|
| SP2 | | |
| LOD | | ADDRESS OF DIVISOR |
| TCX | | TRANSFER ADDRESS TO INDEX REGISTER |
| SP4 | | |
| LOD | | DIVIDEND |
| BRM | FD | |
| SOV | | TEST FOR ERROR (OPTIONAL) |
| BRU | *+7 | NO ERROR, CONTINUE PROGRAM |
| SAZ | | TEST TYPE OF ERROR |
| BRU | | OVERFLOW ERROR EXIT |
| BRU | | UNDERFLOW ERROR EXIT |
5. FD requires 146 bytes of memory plus 14 bytes of temporary storage. The time of execution of FD is from 1,125 to 1,236 cycles depending upon the number of one bits in the quotient.
 6. Floating point divide is implemented by executing the following sequence of operations:
 - a. Save the dividend mantissa, get the divisor mantissa and save it, test the divisor for zero and take error exit if zero.
 - b. Compute and save the sign of the quotient by an exclusive OR operation between the dividend sign and divisor sign.

- c. Compute and save the exponent of the quotient by subtracting the divisor exponent from the dividend exponent.
- d. Initialize the left most byte of the divisor and dividend and the 4-byte quotient. The eighth bit from the right is set in the quotient to indicate the last pass through the divide loop when that bit is in the 32nd place from the left.
- e. The divide is executed by doing the following steps 24 times:
 1. Load the current dividend, shift it left once and subtract the divisor.
 2. Test the sign of the remainder, if negative go to 3, if positive go to 4.
 3. Restore the divisor back into the current dividend and store new dividend. Load the quotient and shift it left once (enter a zero bit for the current divide step) go to step 5.
 4. Store the new dividend, load the quotient, shift it left once and increment it (enter a one bit for the current divide step), continue with step 5.
 5. Store the new quotient and test its most significant bit which will be set on the 24th pass. If not set, return to step one, otherwise continue.
- f. Test for zero quotient and, if found, exit with zero accumulator.
- g. Test if the quotient is normalized, if not shift left one to normalize and set the overflow flag to indicate the operation.
- h. Load the quotient exponent, test the overflow flag indicating the mantissa shift in the previous step. If it is set, decrement the exponent.
- i. Test if exponent underflow or overflow occurred. Take error exit, if the exponent is out of range.
- j. Construct the floating point quotient by assembling the quotient sign, exponent, and mantissa into the floating point format and return to caller.

The error exit taken when exponent overflow or underflow is detected sets the overflow register. If overflow occurred, the largest possible number with the sign of the quotient is returned. If a zero divisor was entered, the largest possible positive number will be returned. In many cases, the error return format will allow the user to continue his program using the approximation returned instead of taking special error exits. The accuracy of FD is 23 bits in the mantissa.

IP — Integer Part of Floating-Point Number

1. IP extracts the integer part of a floating-point number. The fractional part of the number is lost.
2. On entry, the 4-byte accumulator contains the floating-point number. On exit,

mathematical subroutines

the 4-byte accumulator contains the integer portion of the entered number in floating point format. The X register is unchanged, overflow is reset, and precision 4 is set.

3. No error conditions are detected in this routine. Zero is returned if the input number is all fraction.

4. Coding sequence:

SP4

LOD FLOATING POINT
 NUMBER

BRM IP

5. IP is coded in common with FP. The two routines are 110 bytes long plus 2 bytes of temporary storage.

The time for execution of IP is from 124 to 704 cycles. The shortest time is required when the number is all fraction. When the number is all integer, about 190 cycles are required. When the number is part integer and part fraction, the time varies from 314 to 704 cycles depending upon the amount of shifting and normalizing required for execution.

6. The floating-point number entered to the IP routine is separated into the exponent and mantissa parts by calling the SE routine. The exponent is tested to determine if the number is:

- All fraction (zero or negative exponent)
- All integer (exponent greater than 23)
- Part integer and part fraction

If the number is all fraction, zero is returned. If the number is all integer, the number is returned. If the number is part integer and part fraction, the following steps are executed:

- a. Twenty-four (24) is subtracted from the exponent and the result is transferred to the X register.
- b. The mantissa is shifted right and the X register is incremented repeatedly until the X register is zero. This removes the fractional part of the mantissa.
- c. The exponent is set to 24 and is combined with the mantissa to form a floating point number. (FC routine).
- d. The floating-point number is normalized (FN routine), the X register is restored (RX routine), and the routine returns.

7. IP calls the following subroutines; SE, FC, FN, and RX.

FP — Fractional Part of Floating-Point Number

1. FP extracts the fractional part of a floating-point number. The integer part of the number is lost.
2. On entry, the 4-byte accumulator contains the floating-point number. On exit, the 4-byte accumulator contains the fractional portion of the entered number in floating-point format. The X register is unchanged, overflow is reset, and precision 4 is set.

3. No error conditions are detected in this routine. An error is returned if the input number is all integer.

4. Coding sequence:.

SP4

LOD FLOATING POINT
NUMBER

BRM FP

5. FP is coded in common with IP. The two routines are 110 bytes long plus 2 bytes of temporary storage.

The time for execution of FP is from 127 to 648 cycles. The shortest time is required when the number is all integer. When the number is all fraction, approximately 180 cycles are required. When the number is part integer and part fraction, the time varies from 264 to 648 cycles depending upon the amount of shifting and normalizing required for execution.

6. The floating point number entered to the FP routine is separated into the exponent and mantissa parts by calling the SE routine. The exponent is tested to determine if the number is:

1. All fraction (zero or negative exponent)
2. All integer (exponent greater than 23)
3. Part integer and part fraction.

If the number is all integer, zero is returned. If the number is all fraction, the number is returned. If the number is

part integer and part fraction, the following steps are executed:

- a. The exponent is transferred to the X register.
- b. The mantissa is shifted left and the X register is decremented repeatedly until the X register is zero. This removes the integer part of the mantissa.
- c. The exponent is zero and is combined with the mantissa to form a floating point number. (FC routine.)
- d. The floating point number is normalized (RN routine), the register is restored (RX routine), and the routine returns.

7. IP calls the following subroutines: SE, FC, FN, and RX.

SQ – Floating-Point Square Root

1. The SQ routine calculates the square root of a floating-point number using Newton's iterative method. The number must be positive.
2. On entry, the 4-byte accumulator contains the floating-point number. On exit, the 4-byte accumulator contains the floating-point square root of the entered number. The X register is unchanged, overflow is reset, and precision 4 is set.
3. If the number entered is negative, that number is returned and overflow is set.

mathematical subroutines

4. Coding sequence:

SP4

LOD NUMBER TO BE CON-
 VERTED

BRM SQ

SOVN TEST FOR ERROR
 RETURN

BRU ER TAKE ERROR EXIT

5. SQ requires 105 bytes of core plus 12 bytes of temporary storage. The time for execution of SQ is from approximately 5090 cycles to 5455 cycles, depending on the time required by the subroutines called.

6. The square root is calculated by using Newton's iterative method:

$$X_{i+1} = 1/2 (X_i + N/X_i); \quad i = 1 \text{ to } 3$$

$$X_1 = 1/2 (N + 0.875)$$

Where:

N = the input number mantissa ($1 < N \leq 0.25$)

X_i = the previous approximation

i = index from 1 to 3

X_1 = the first approximation

This method is implemented in the SQ routine in the following manner:

a. The floating point number entered (N) is separated into exponent and

mantissa. The number is tested and the error exit is taken if the number is negative.

- b. The exponent is tested. If the exponent is odd, it is decremented to make it even and the mantissa is right shifted once to compensate.
- c. The exponent is divided by 2 (shifted right once) and stored as the exponent for the square root of N.
- d. The mantissa is now between 0.25 and 1 and is stored as N and A for the iterative loop.
- e. The constant for the first approximation of N/A (0.875) is loaded into the accumulator and the loop is entered at step 2. This causes the result of step 2 to be $0.875 + N$, the first approximation.
- f. Loop to execute Newton's iteration:
 1. Load the index register with the address of A, load N into the accumulator, and call FD. The result is N/A .
 2. Call FA. The result is $N/A + A$.
 3. Divide by 2 by decrementing the exponent once. The result is $(N/A + A) 1/2$. Save this result as A for the next iteration or for the final result.
 4. Increment and test the loop count. If the count is zero (fourth pass) continue, otherwise loop back to step 1.

- g. The exponent of the square root calculated in step c is combined with the result of the loop which is the mantissa of the square root of N. This floating point square root is then returned. The accuracy of SQ is 23 significant binary bits in the mantissa.
7. Subroutines called: SE, RX, FD, FA. FN is called indirectly through FA.
5. EX requires 209 bytes of memory plus 10 bytes of temporary storage. The time of execution of EX is from 9,440 to 9,950 cycles depending if X is all fraction or if X is part integer.
6. Floating point exponential function is computed by using an approximation method by W. J. Cody and A. Ralston (*A Note on Computing Approximations to the Exponential Function*, Communications of the ACM, Vol. 1, pp. 53-55, January 1967):

EX — Floating Point Exponential Function

1. EX computes the exponential value of the floating-point number X entered (e^X).
2. On entry, the 4-byte accumulator contains the floating-point exponent X. On exit, the 4-byte accumulator contains the floating-point exponential value (e^X). The X register is unchanged, overflow is reset, and precision 4 is set.
3. An error is indicated by a return with the overflow register set when the input value of X is too large, causing overflow, or too small, causing underflow. The approximate range of X is (-44 to 44). If overflow occurs, the largest positive floating point number is returned in the accumulator. If underflow occurs, zero is returned in the accumulator.

4. Coding sequence:

SP4	
LOD	VALUE OF X
BRM	EX
SOVN	TEST FOR EXIT
BRU	ERROR EXIT

By using the relationship

$$X \log_2 e = n + r, n \text{ integer} \quad -1 < r < 1$$

the exponential function becomes

$$e^X = 2^n 2^r = 2^n (e^{\ln 2})^r = 2^n (e^Y)$$

$$\text{where } y = (1/2)(r)$$

The 2^n factor is implemented by a change of exponent. The range of y is $(-1/2, 1/2)$ and e^Y is approximated by the rational function

$$R_3(y) = 1 + \frac{2y}{S_3(y) - y}$$

where the rational function

$$S_3(y) = A_0 - \frac{A_1}{B_1 + y^2}$$

The accuracy of the resulting answer would be exact over the floating-point range. However, the floating-point multiplications and divisions required to

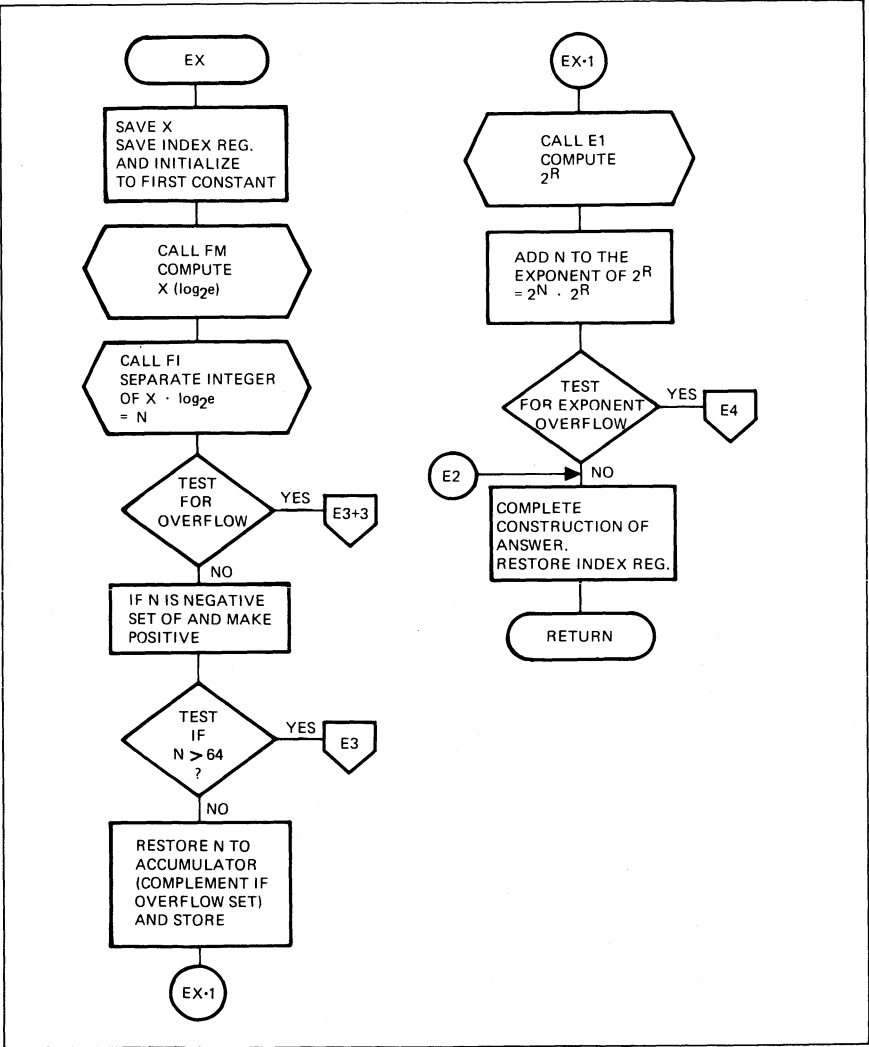


Figure 19-4. Flowchart of EX (Sheet 1)

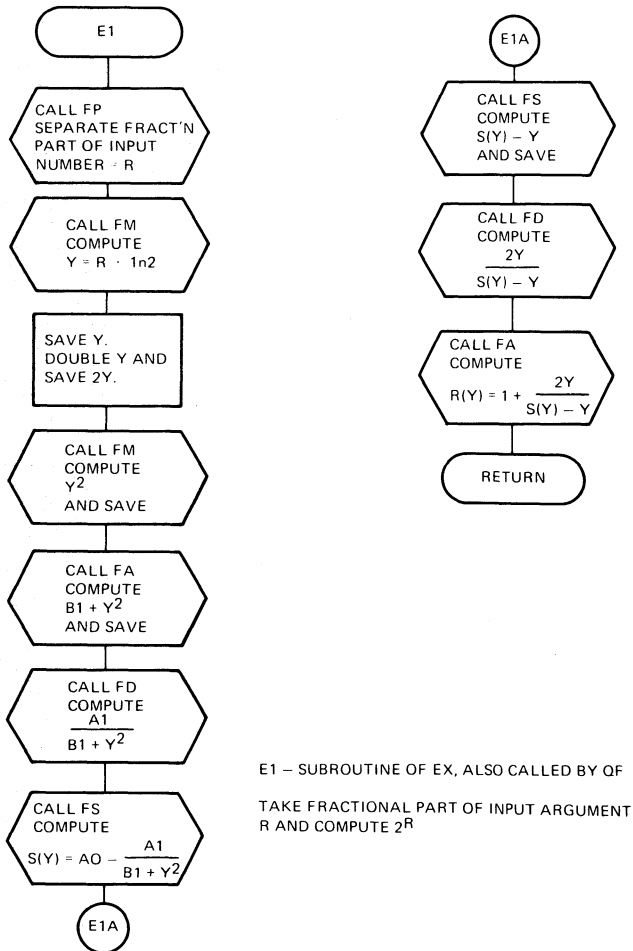


Figure 19-4. Flowchart of EX (Sheet 2)

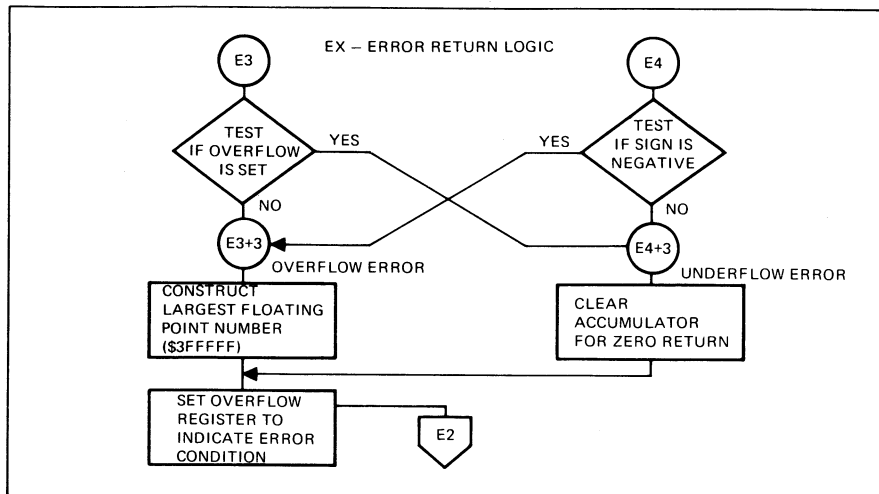


Figure 19-4. Flowchart of EX (Sheet 3)

compute the result degrade the accuracy to 22 significant bits in the mantissa.

The method of implementation of the equations is detailed in Figure 19-4.

- Subroutines called: FI, FA, FS, FM, FD, FP, RX, SE, FC, and FN are called indirectly.

exit, the 4-byte accumulator contains the floating-point natural logarithm of X. The index register is unchanged, overflow is reset, and precision 4 is set.

- An error is indicated by a return with the overflow register set when the input value of X is negative or zero. If a negative number was entered, zero is returned. If zero is entered, the largest negative number is returned.

LN – Floating Point Natural Logarithm

- LN computes the natural logarithm of the floating-point number X entered.
- On entry the 4-byte accumulator contains the floating-point number X. On

- Coding sequence:

SP4	
LOD	VALUE OF X
BRM	LN

SOVN TEST FOR ERROR
BRU ERROR EXIT

- LN requires 173 bytes of memory plus 10 bytes of temporary storage. The time of execution of LN is approximately 10,550 cycles, depending upon the variance of time due to the many other routines called by LN. The time calculation takes into account the particular parameters involved when other routines are called.
- Floating point natural logarithm function is computed by reducing the range of the input argument and using an approximation by Lyusternik et. al., Handbook for Computing Elementary Functions, page 66:

The relationships used are:

$$(1) \ln X = \ln(2^n y) = \ln 2^n + \ln y$$

$$(2) \ln X = n (\ln 2) + \ln y; 1/2 \leq |y| < 1$$

$$(3) \ln y = -1/2 \ln 2 + A_1 U + A_3 U^3 + A_5 U^5$$

$$(4) U = \frac{y - \sqrt{2/2}}{y + \sqrt{2/2}}$$

The implementation of these equations is detailed in Figure 19-5.

The accuracy of the answer is $\pm 0.12 \times 10^{-7}$

- Subroutines called: SE, IF, FM, FA, FS, FD, RX; FN is called indirectly

FE — Floating Point Exponentiation

- FE computes $A^{**}B$ where A and B are floating-point numbers.
- On entry, the 4-byte accumulator contains the floating-point base A, and the X register contains the address of the floating-point exponent B. On exit, the 4-byte accumulator contains the result $A^{**}B$. The X register is unchanged, overflow is reset, and precision 4 is set.
- An error is indicated by a return with the overflow register set. If the base A was negative, the accumulator will contain all ones on return. If the base A was zero, zero will be returned in the accumulator. If the result $A^{**}B$ is too large or too small, the accumulator will contain the largest possible number or zero on return.

- Coding sequence:

SP2	
LOD	ADDRESS OF EX- PONENT B
TCX	
SP4	
LOD	BASE A
BRM	FE
SOVN	
BRU	ERROR EXIT

- FE requires 23 bytes of memory. The time of execution of FE is approximately 21,200 cycles, depending upon the variances of time due to the routines called.

FLOWCHART OF LN

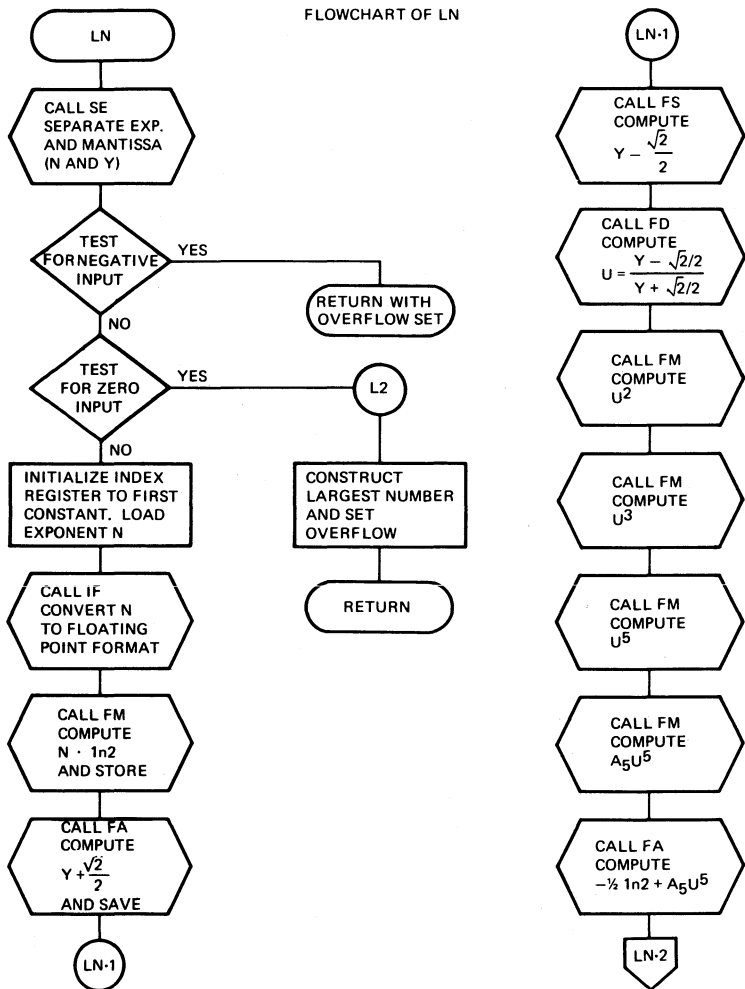
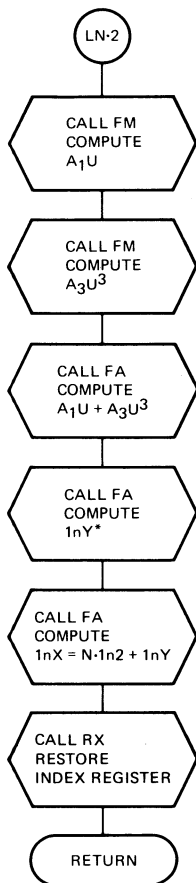


Figure 19-5. Flowchart of LN (Sheet 1)



$$*1nY = -\frac{1}{2}1n2 + A_1U + A_3U^3 + A_5U^5$$

Figure 19-5. Flowchart of LN (Sheet 2)

mathematical subroutines

6. Floating point exponentiation ($A ** B$) is computed by using the equality

$$A^B = e^{\ln(A ** B)} = e^{B \ln A}$$

The expression on the right is solved by calling the routines LN, FM, and EX in that sequence. No additional manipulation is required since the calling sequence for FE meets the requirements for the other routines. The return from LN is tested for zero or negative value of A. If overflow occurs during execution of FM or EX, the error returns from those routines satisfy the error return for FE. The accuracy of FE is 21 significant bits in the mantissa.

7. Subroutines called directly are: LN, FM, EX; subroutines called indirectly are: RX, SE, FC, FN, IF, FI, FA, FS, FD and FP.

QF – Decimal to Floating Point Conversion

1. QF converts a decimal fraction and exponent in memory into a four-byte floating-point binary number. The mantissa may be from one to ten signed decimal digits, and the exponent may be one or two signed decimal digits.
2. On entry, the X register must contain the address of the Header Cell of the number to be converted. At the address given in X, memory must contain the number to be converted in the specified format.
3. If an error condition exists, overflow is set. If the number was too large or too small, the largest floating-point number

or zero is returned. If the format is incorrect, the 4-byte accumulator will contain all ones.

4. Coding sequence

SP2

LOD	ADDRESS OF DECIMAL NUMBER
TCX	TRANSFER ADDRESS TO INDEX
BRM	QF
SOVN	TEST FOR ERROR
BRU	BRANCH TO ERROR ROUTINE

5. QF requires 165 bytes of memory plus 6 bytes of temporary storage. The time of execution of QF is approximately 12,100 cycles, depending upon the number of digits entered, and the variance of the subroutines called.

6. Decimal to floating point conversion performs the conversion using the following relationships:

$$X \cdot 10^Y = X \cdot 2^Y (\text{LOG}_{210}) = X \cdot 2^{(N+R)}$$

$$N = \text{Integer part of } Y (\text{LOG}_{210})$$

$$R = \text{Fractional part of } Y (\text{LOG}_{210})$$

$$X \cdot 10^Y = X \cdot 2^N \cdot 2^R$$

The 2^N value is implemented by an exponent adjustment. The 2^R value is computed in the subroutine of the exponential function EX.

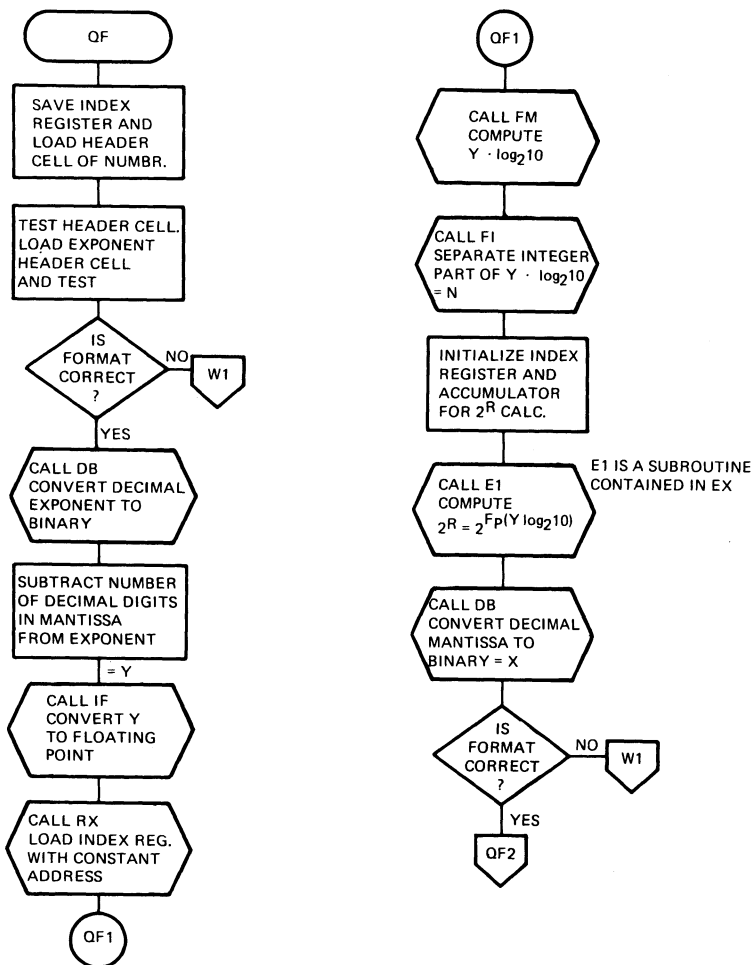


Figure 19-6. Flowchart of QF (Sheet 1)

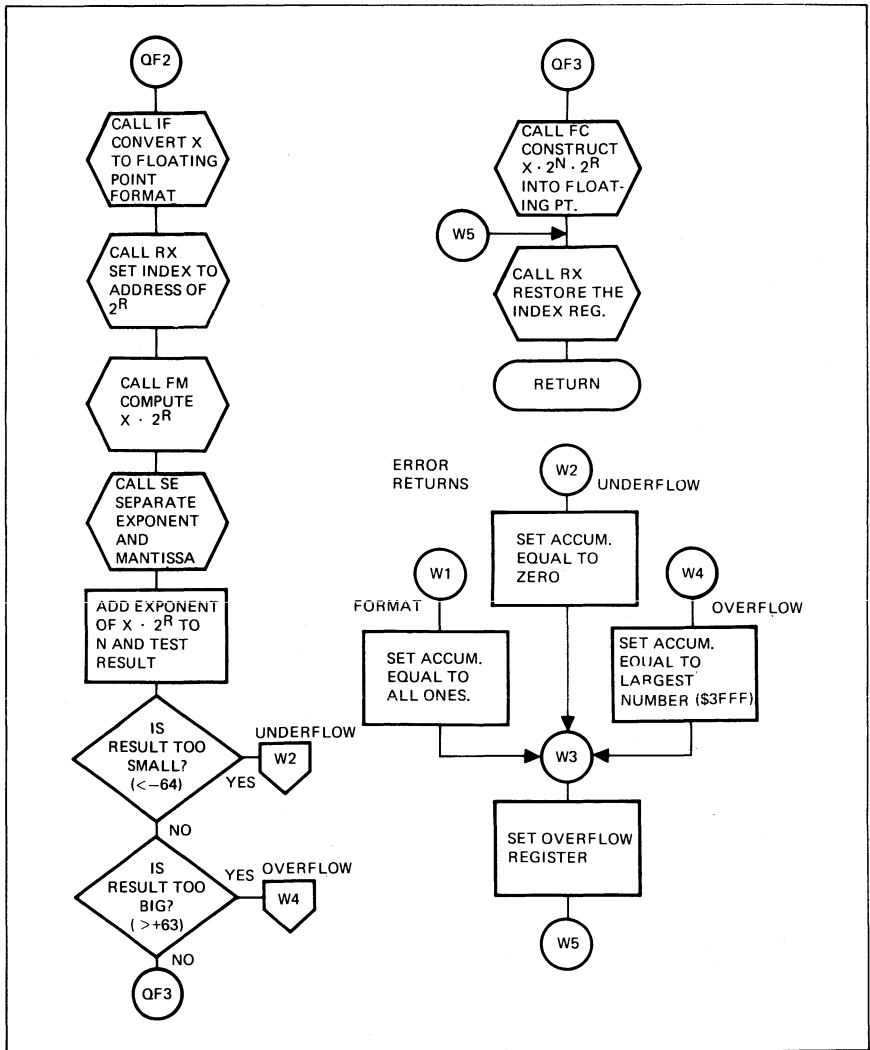


Figure 19-6. Flowchart of QF (Sheet 2)

The method of implementing these equations is detailed in Figure 19-6.

- Subroutines called directly are RX, SE, IF, FI, FC, FM, EX, and DB; subroutines called indirectly are FN, FA, FS, FD, and FP.

FQ — Floating Point Binary to Decimal Conversion

- FQ converts a four-byte binary floating-point number into a decimal fraction and exponent in memory. The decimal number is seven signed decimal digits followed by two signed decimal digits representing the exponent.
- On entry, the 4-byte accumulator contains the floating point number to be converted. The X register contains the address of the area in memory in which the decimal number is to be placed. The area must be 11 bytes.

On exit, X is unchanged, the C register has the same contents as the X register, the A register is unspecified, precision 4 is set, and overflow is reset. The area in memory specified contains the decimal number.

- No error conditions are detected by FQ.
- Coding sequence:

SP2	
LOD	ADDRESS OF OUTPUT AREA
TCX	TRANSFER TO INDEX REGISTER

SP4

LOD NUMBER TO BE CONVERTED

BRM FQ

- FQ requires 138 bytes of memory plus 8 bytes of temporary storage. The time of execution of FQ is approximately 15,200 cycles, depending upon the variance in time of the subroutines called.
- The floating-point binary-to-decimal conversion is accomplished by evaluating the following equations:

$$A \cdot 2^N = X \cdot 10^Y$$

$$Y^1 = N \cdot \log_{10} 2 = Y^1_{fp} + Y^1_{ip}$$

$$Y = Y^1_{ip}$$

$$X = A \cdot C^{[Y^1_{fp}(\log_{10})]}$$

where:

A = binary mantissa (fraction)

N = binary exponent

X = decimal mantissa (fraction)

Y = decimal exponent
fp denotes fractional part
ip denotes integer part

The method of implementing these equations is detailed in Figure 19-7.

The accuracy of the conversion is plus or minus 3 in the least significant decimal digit. When the decimal number is

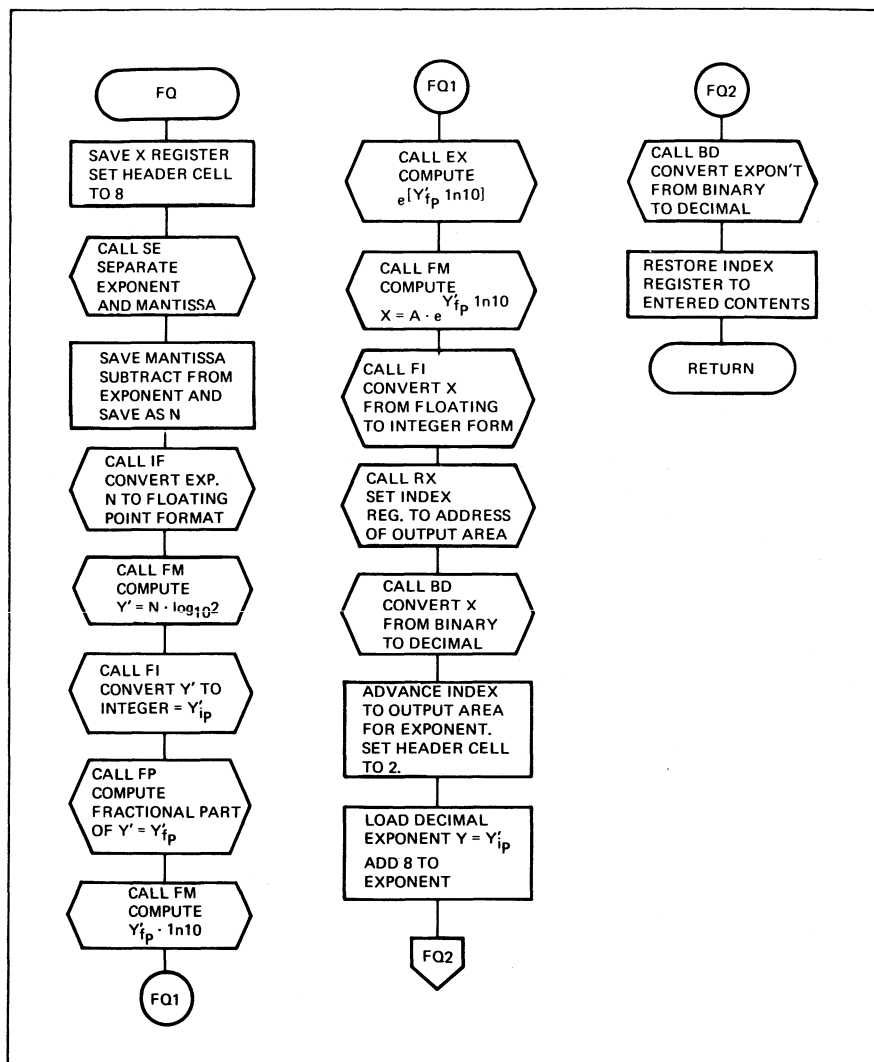


Figure 19-7. Flowchart of FQ

between 0.839 and 1.0, there is a leading zero in the decimal mantissa with the exponent adjusted accordingly. (e.g; 93.5 is expressed as 0.0935 E3).

7. Subroutines called directly are RX, SE, FC, IF, FI, FM, FP, and EX; subroutines called indirectly are FN, FA, FS, and FD.

SI – Floating Point Sine Function

1. SI computes the sine of the floating-point number X entered. The function is approximated by a truncated Taylor's series.
2. On entry, the 4-byte accumulator contains the floating-point number X . On exit, the accumulator contains the floating-point sine of X . The index register is unchanged, overflow is reset, and precision 4 is set.
3. No error conditions are detected in the SI routine.
4. Coding sequence:

SP4

LOD

VALUE OF X

BRM

SI

5. SI requires 206 bytes of memory plus 10 bytes of temporary storage. The time of execution of SI is approximately 13,250 cycles if the number entered is less than 2π , or approximately 16,100 cycles if the number entered is 2π or greater. The time varies due to the variance in the execution time of the subroutines called.

6. The sine of X is calculated using the equation:

$$\sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!}$$

for X in the range $(0 \leq X < \frac{\pi}{2})$

The following equations are used to reduce the number X down to the proper range:

- $\sin(-x) = -\sin(x)$
- $\sin(x + 2\pi) = \sin(x)$, $\pi = \text{INTEGER}$
- $\sin(\pi + x) = -\sin(x)$
- $\sin(\pi - x) = \sin(x)$

If X is greater than 2π , X is divided by 2 and the fractional part of the resulting quotient is multiplied by 2π . The user is cautioned that if X is much larger than 2π , the accuracy of the answer is reduced because the fractional part of the quotient does not have the full number of significant bits and therefore the value of X less than 2 contains a rounding error.

The accuracy of SI when X is less than 2π , is $+4.10^{-7}$.

The details of the implementation of the above equations are presented in Figure 19-8.

7. Subroutines called directly are RX, FA, FS, FM, FD, and FP; subroutines called indirectly are SE, FC, and FN.

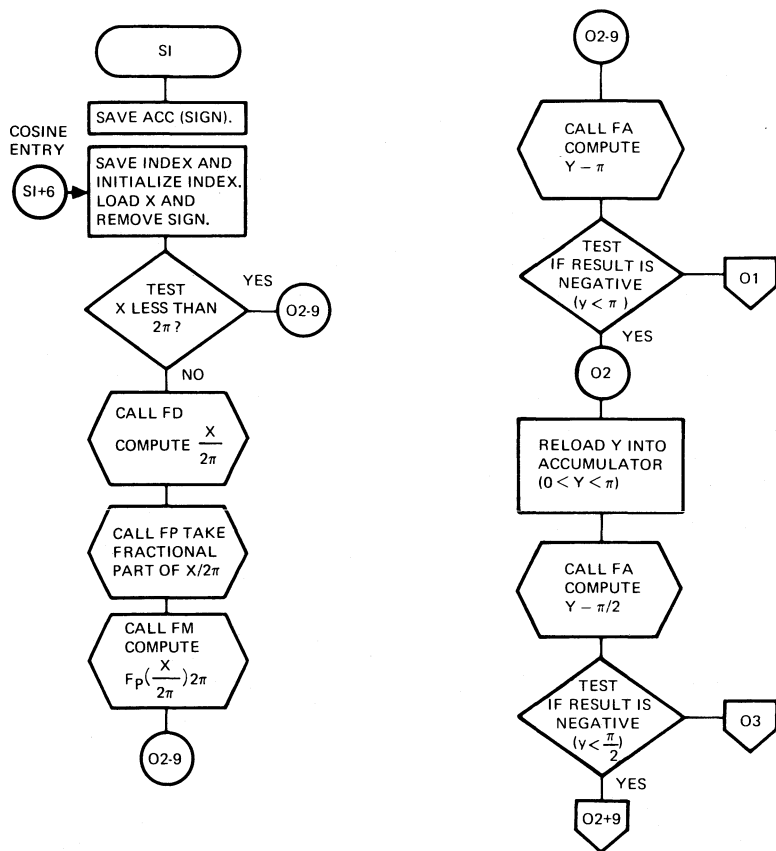


Figure 19-8. Flowchart of SI (Sheet 1)

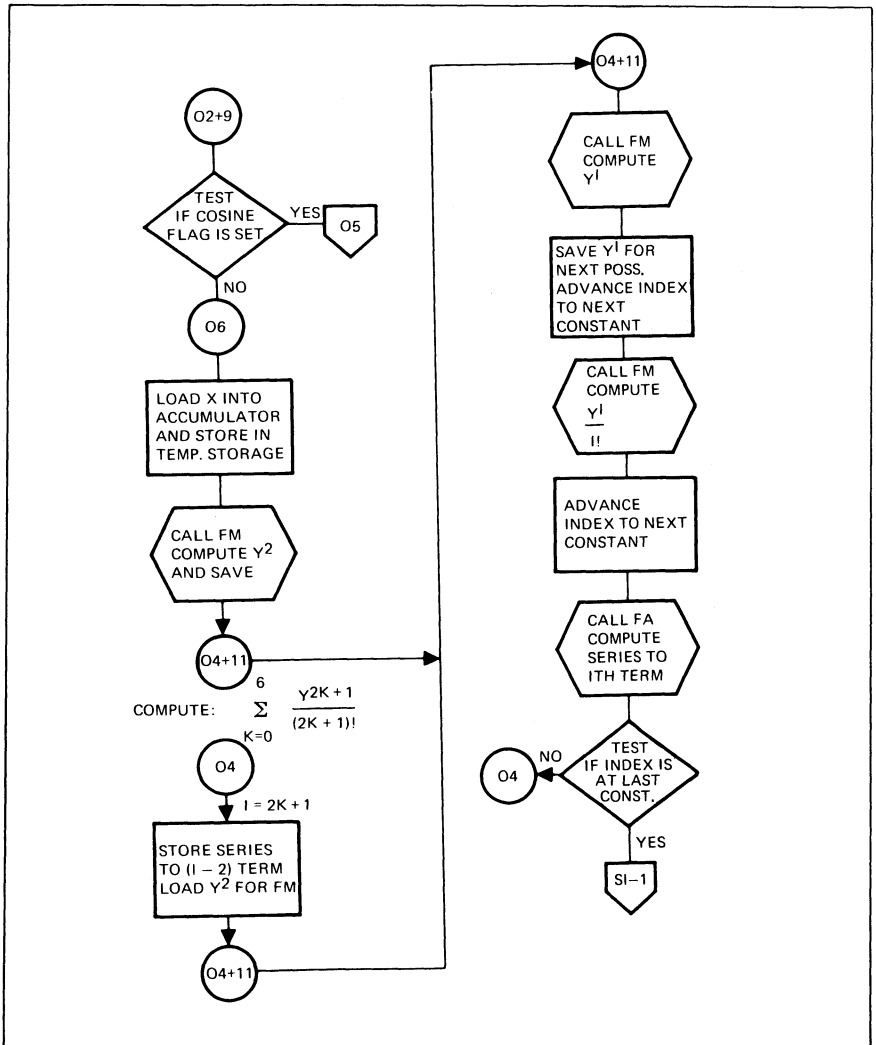


Figure 19-8. Flowchart of SI (Sheet 2)

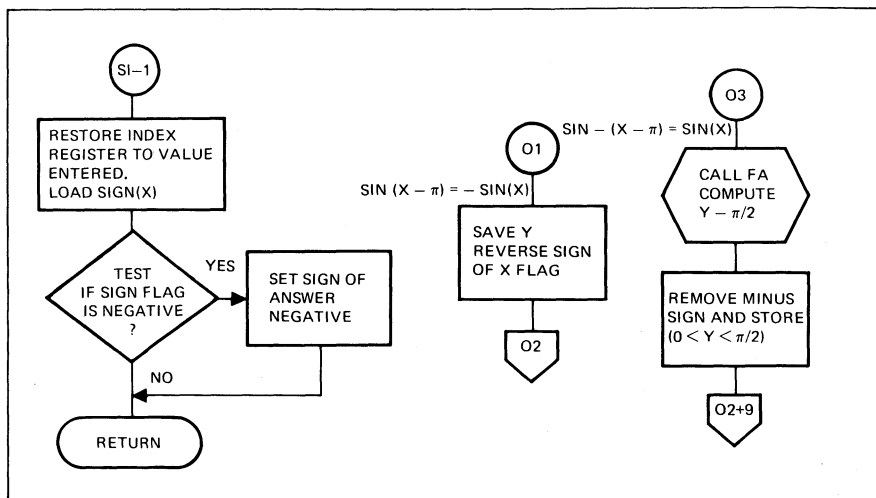


Figure 19-8. Flowchart of SI (Sheet 3)

CO — Floating Point Cosine Function

- CO computes the cosine of the floating-point number X entered. The function is approximated by a truncated Taylor's series.
- On entry, the 4-byte accumulator contains the floating-point number X. On exit, the accumulator contains the floating-point cosine of X. The index register is unchanged, overflow is reset, and precision 4 is set.
- No error conditions are detected in the CO routine.

4. Coding sequence:

SP4

LOD

BRM

CO

VALUE OF X

- CO requires 27 bytes of memory. The time of execution of CO is approximately 13,700 cycles if the number entered is less than 2π or approximately 16,500 cycles if the number entered is 2π or greater.
- The cosine of X is calculated using the relation:

$$\cos(X) = \sin\left(\frac{\pi}{2} - X\right)$$

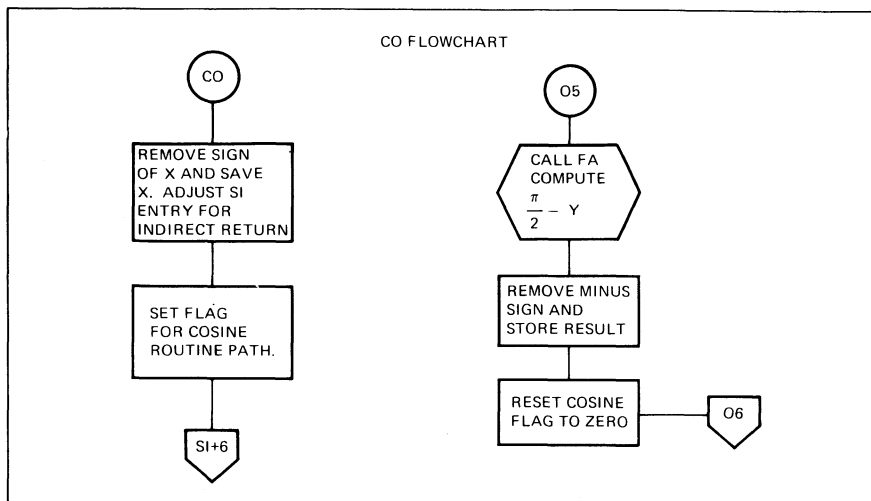


Figure 19-9. Flowchart of CO

The CO routine sets a flag and enters the SI routine. After the SI routine reduces the range of X to the proper interval (0, $\pi/2$), the cosine coding is entered when the cosine flag is set. The cosine coding subtracts $\pi/2$ from the argument, removes the minus sign from the result, resets the cosine flag, and returns to the sine routine.

The cautions and accuracy of the sine routine apply to the cosine routine. The flowchart for the CO function is shown in Figure 19-9.

7. Subroutines called directly are SI and FS; subroutines called indirectly through SI are RX, SE, FC, FN, FA, FM, FD, and FP.

AT — Floating Point Arctangent

1. AT computes the arctangent of a floating-point number.
2. On entry, the 4-byte accumulator contains the floating point number X. On exit, the 4-byte accumulator contains the arctangent of X. The index register is unchanged, overflow is reset, and precision 4 is set.
3. No error conditions are detected by AT.
4. Coding sequence:

SP4

LOD

BRM

VALUE OF X

AT

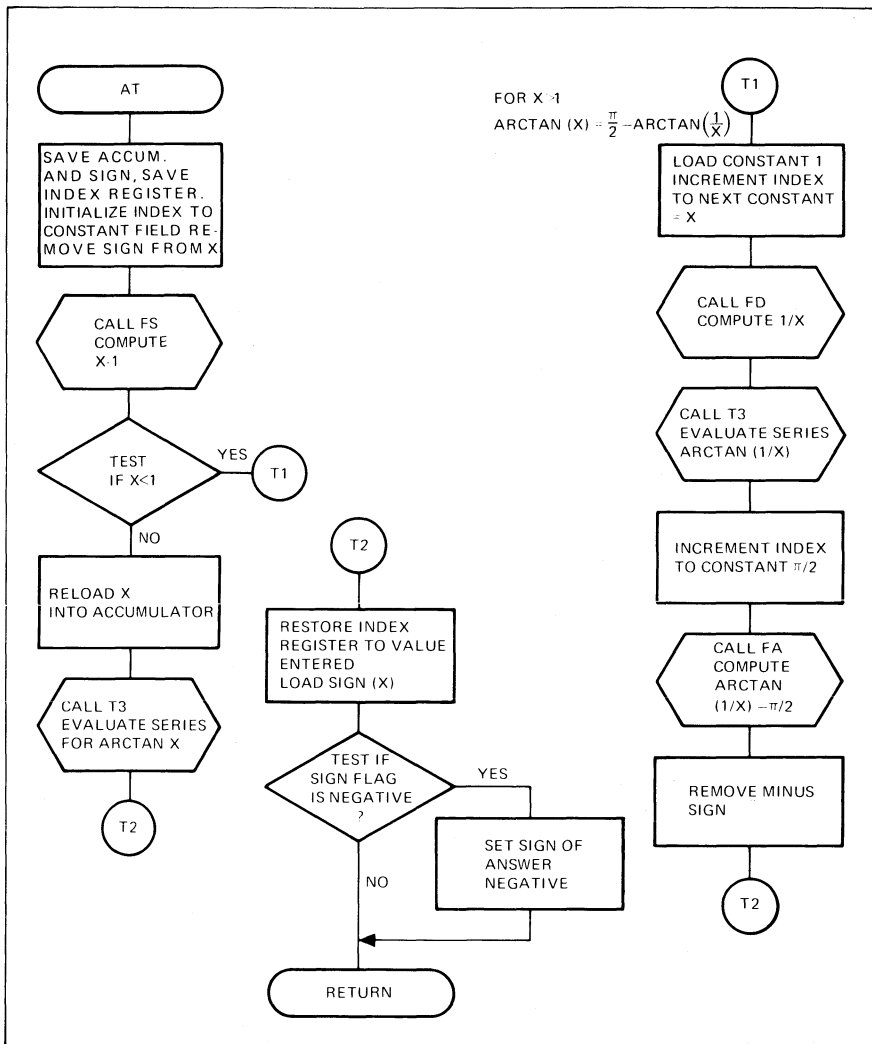


Figure 19-10. Flowchart of AT (Sheet 1)

AT SUBROUTINE = COMPUTE $\sum_{K=0}^6 C_{2K+1} X^{2K+1}$

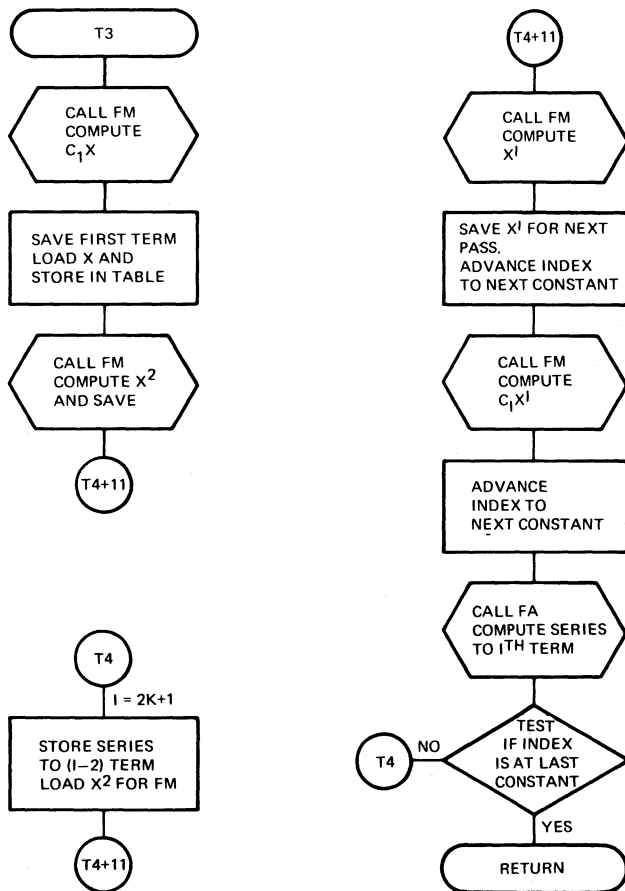


Figure 19-10. Flowchart of AT (Sheet 2)

5. AT requires 203 bytes of core plus 6 bytes of temporary storage. The time for execution of AT is approximately 18,000 cycles.

6. The arctangent of X is calculated by evaluating the following approximation by Hastings, *Approximations for Digital Computers*, page 136:

$$\text{ARCTAN } X \approx \sum_{k=0}^6 C_{2k+1} X^{2k+1}$$

for $0 < X < 1$

The following formulas are used to reduce the range of X to the proper interval; for X negative;

$$\text{ARCTAN } (-X) = -\text{ARCTAN } (X)$$

for X greater than 1;

$$\text{ARCTAN } X = \frac{\pi}{2} - \text{ARCTAN } \left(\frac{1}{X} \right)$$

The implementation of the above equations is detailed in Figure 19-10.

The accuracy of AT is $\pm 0.4 \times 10^{-7}$.

7. Subroutines called directly are FA, FS, FM and FD; subroutines called indirectly by FA are RS, SE, and FN.

SECTION 20 — SOURCE TAPE CORRECTION (STC) PROGRAM

Identification No. 92A0203-005

Features:

- Provides the ability to ADD, DELETE, CHANGE, LIST, and PUNCH any number of source lines
- Buffer capacity is 127 lines and 3000 characters
- Lists and updates decimal line numbers of lines in buffer
- Provides free format source output for Symbolic Assembler

The Source Tape Correction Program (STC) provides the ability to edit a symbolic source tape using the teletype as the interface device. The program allows the user to ADD, DELETE, CHANGE, LIST, and PUNCH any number of source lines up to the entire contents of the 127-line, 3,000-character buffer.

The STC program eliminates the tedious task of correcting symbolic source tapes off line. An appreciable amount of time and labor is saved, and the ease of operation of the STC allows any user to quickly take advantage of the program.

The STC program occupies 986 bytes of core and allows approximately 3,000 bytes of core for the buffer. In its basic configura-

tion, the STC is designed to operate with the minimum Varian 520/i system of 4K memory and a Model 33TC teletype. The start address of the STC is location \$0D.

The STC may be adapted to larger memory configurations of the Varian 520/i by altering the END buffer limit. The value of END is initially set at -\$FAO to prevent the Binary Loader from being overlaid. The limit may be set, however, to any value in memory desired by the user. The 2's complement of the desired value must be entered for correct operation. The END counter is located at \$0B.

The start address of the buffer, which is set at \$3EO, may also be changed by the user to allow other programs to remain in core. This is done by altering the contents of the BO pointer located at zero.

Operating the STC Program

The STC program responds to a series of commands entered by the teletypewriter. These commands are summarized in Figure 20.1 and detailed in the following sections.

First step in operating the STC program is to load the STC object tape into the computer with the loader mode set to non-zero (load and transfer). Loaded in memory, the program outputs a carriage

INPUT COMMANDS

R.	Read input tape until full buffer or end of tape
R,N.	Read N Lines from the input tape

EDITING COMMANDS

A.	Append to the buffer from the keyboard
DN.	Delete line N from the buffer
DN,M.	Delete lines N through M from the buffer
IN.	Insert before line N from the keyboard
CN.	Change line N (delete then insert)

OUTPUT COMMANDS

LN.	List (Punch) line N
LN,M.	List (Punch) lines N through M.
P.	Punch leader/trailer code

OTHER COMMANDS

H —	Give current value of Here line
E —	Give current value of End line
←	Cancel command
←	Cancel next line input to buffer
"CONTROL-D"	Return to command mode

Figure 20-1. Summary of STC Commands

return and line feed and awaits a typed command from the keyboard.

The user then enters his program tape in the tape reader, sets Sense Switch 1 to suppress simultaneous listing, and enters an Input Command. The STC reads a section of the source program into the buffer and outputs a carriage return and line feed upon completion. The user then begins editing, using the commands described below.

If an incorrect command is entered, STC responds by outputting a question mark, followed by a carriage return and line feed, and then awaits another command. If the user wants to cancel a command before completing it, he inputs a back arrow (←) and the STC program ignores the command, outputs a carriage return and line feed, and awaits another command. If the buffer is overrun, STC outputs 0?, followed by a carriage return and line feed and awaits further commands.

The STC program may also be used for the initial generation of a source tape (Pass I of the Symbolic Assembler operation). When used in this manner, a line feed is generated automatically each time a carriage return is input. The STC keeps continuous track of the total number of lines in the buffer as well as the number of the present line being processed, and allows access to any line or lines indexed by these numbers.

The lines are consecutively numbered in a decimal count, with the first line in the buffer as number one. The numbers are continuously updated as each change is made in the program. Whenever the user

requests a listing of the current form of his program, each line of the listing is preceded by its line number.

Input Commands

Input Commands are used to enter the source program into the buffer for correction. Sense Switch 1 controls the listing of the source tape while it is being read into the buffer. If it is not set, the source tape is listed by the teletype, a character at a time, as the tape is input. In this mode, the tape is read at a rate of five characters per second. If Sense Switch 1 is set, no listing occurs, and the tape is read at a rate of ten characters per second. It is recommended that the user enter the tape with the sense switch set to suppress listing, and then, if a listing is desired, command the listing as directed by the Output Commands. In this way, the listing is obtained with the buffer line numbers aiding the correction procedure.

The Input Commands are designed to allow the user to start input into the STC buffer at any point in his program. Portions of a source tape which are correct may be reproduced directly off-line. When areas which require correction are encountered, they may be entered into the buffer and corrected.

To process a program which exceeds the buffer capacity, the user may edit the source tape in sections by filling the buffer, correcting that segment of the program, punching out the corrected segment, then refilling the buffer with the next segment.

source tape correction (STC) program

R. Read source tape

Read source tape from the tape reader until the end of the tape (blank trailer) or a full buffer condition is reached.

The source tape is read into the buffer a line at a time. Each line is separated by a carriage return and line feed (in that order) and a 16 bit address of the next line. The end of each line is indicated by the carriage return character (\$8D). If the source tape contains lines which end with a line feed and then a carriage return, the STC enters the line feed into the buffer as a program character, detects the carriage return, then generates a line feed following the carriage return.

A full buffer condition is reached when either 127 lines or 300 characters have been entered into the buffer.

R, N. Read N lines

Read N lines from the tape reader, stop after N lines have been entered, or when the tape ends, or when a full buffer condition is reached, whichever comes first.

This command is identical in operation to R. described above, except that no more than N lines are input. The digital number N may not exceed 127. If a larger number is designated, it is read as 127.

Editing Commands

Editing Commands are used to change the source program after it has been read into the buffer.

A. Append Input

Append input from the keyboard to the buffer (after the last line in the buffer) until the "Control-D" key combination is entered (CON-D).

Each line must be ended by depressing the carriage return key (CR). When the CR is entered, a line feed is automatically generated. Lines may be input into the buffer until a full buffer condition is reached.

Once the command is entered, only source lines may be entered until the CON-D code is input. The CON-D code is recognized only when it immediately follows a CR entry.

If the text is entered incorrectly, a line may be deleted prior to ending it with the CR character by entering the back-arrow (←) character.

When the back arrow character is entered, the entire current line being entered is deleted, a carriage return and a line feed is output, and new lines of text or CON-D may be input.

DN. Delete line N

DN, M Delete lines N through M

Delete lines N, or lines N through M, from the buffer. The lines indicated are removed from the buffer and the reference line numbers are changed to indicate the new condition of the buffer.

The actual memory occupied by the deleted lines is not recovered. The line addresses are altered to skip over the

deleted lines, and the total line counter is decremented accordingly. This means that no additional character space is gained, making it possible to overflow the buffer by deleting and inserting or changing lines. This possibility is only approached when a very large program is entered into the buffer.

IN. Insert typed-in-text

Insert typed-in-text before line N until the "Control-D" key combination is entered (CON-D). Any number of lines may be inserted.

Lines entered using the IN. command appear on the source listing before line N. All of the affected line numbers are changed, and the total line count is increased by the number of lines entered.

Insertion is implemented by adding the inserted lines to the end of the buffer, and changing the next line-addresses which are between lines in the buffer to cause the inserted lines to appear in the proper sequence.

CN. Change line N

CN, M. Change lines N through M

Change line N or lines N through M. The Change command is a combination of the Delete and Insert commands. The line or lines specified in the Change command are deleted, and then any number of lines may be inserted until the CON-D character is input.

Output Commands

Output Commands are used to list or punch the source program from the buffer using the teletype.

LN. List line N

LN, M. List lines N through M

List line N or lines N through M with Sense Switch 2 not set; the requested line or lines are output to the teletype. Each line is preceded by its current line number. When the user wants to output all or part of his source program to paper tape, Sense Switch 2 must be set before the List command is entered. The command initiates three actions. First, the computer halts to allow the user to turn on the punch. Second, after the user depresses the RUN button on the computer, the specified lines on the teletype (but not the reference line numbers) are output. Third, after the requested lines are output, the computer again halts to allow the user to turn off the punch.

After the punch is turned off, the user depresses the RUN button on the computer to prepare the STC to accept further commands. Sense Switch 2 should be re-set at this time unless additional lines are to be punched. The halting of the computer to turn the punch on and off prevents extraneous characters to be output to the punch tape.

P. Punch leader or trailer

Punch blank leader or trailer tape. When this command is entered, the computer

source tape correction (STC) program

halts. The user turns on the punch and depresses the RUN button on the computer. Eight inches of blank tape are output and the computer again halts. The user turns the punch off and depresses the RUN button on the computer and the STC again accepts further commands.

Line Number References

The line numbers to which N and N, M refer are consecutive digital numbers referring to the current order of the lines in the buffer. In addition to decimal numbers 1 to 127, symbolic numbers E and H are used. E (End) represents the last line in the buffer and is therefore the total number of lines in the buffer. H (Here) represents the last line that has been acted upon. For example, after a user executes the command L7., the value of H is 7.

The E and H symbols may be modified by addition or subtraction. For example, if H is 7 and E is 50, the command L10,40. may also be entered as LH+3, E-10. Similarly, LH, E. is equivalent to L7,50. L1,E. will

cause the entire buffer to be listed. A number greater than E will not be accepted. N may not be larger than M.

Other Commands

The present value of the E and H symbols (see previous paragraphs) may always be known by typing the symbol followed by a dash. The teletype responds by typing the present value and then awaiting further commands.

E-List E

List the number of the end line in the buffer.

H-List H

List the number of the "Here" line. The "Here" line is the line which was last acted upon by the STC program. For example, after executing the command L3,7., the value of H is 7. If the command I9. is entered and the user types in 4 lines, then CON-D, the value of H would be 13.

SECTION 21 — GENERAL DEBUGGING (GAID)

Identification No. 92A0203-004

Features:

- Provides on-line debugging capability to the user
- Performs required addressing mode changes and pointer construction automatically during patch operations
- Provides variable size and multiple location patching areas
- Accepts both paper tape and keyboard commands

5. Execute the Binary Loader to input object tape into the computer.
6. Construct patches for a program in a variable size and location patch area.
7. Perform program run time displays of variable areas of core.
8. Remove a patch or run time display branch from a program and restore the previous instructions.

The GAID is designed to provide the user with a high level of interaction with his program during debugging operations. The patching and dumping capabilities free the user from the mechanics of constructing the actual sequences involved. The automation of these procedures saves the user an appreciable amount of time, considering the complexities of the variable length command and multiple addressing mode features of the Varian 520/i. All required addressing mode changes and pointer construction are done automatically, thereby freeing the user from this responsibility. In addition, the user has control over the location and size of the patch area and the pointer area. Several different areas may be specified during a debugging operation.

Another feature of GAID is the ability to command it from paper tape as well as from keyboard. This enables a user to prepare a

The Varian 520/i General Debugging Aid (GAID) is a system of routines which provide an on-line debugging capability to the user. GAID allows the user to perform the following functions:

1. Display and alter variable fields of memory.
2. Branch to a program with all conditions of the system specified.
3. Branch from a program to GAID and save all conditions at the time of the branch.
4. Produce a loadable punched paper object tape of desired areas of core.

general debugging (GAID)

program debugging sequence off-line before the computer is available. The user may then load his program and ready his tape of commands for GAID when the computer becomes available. After execution, he may analyze the results of his efforts (again off-line) by reading the teletype listing produced. If additional debugging is required, the user may enter through the tape reader previous command tapes to set up his program patches and changes instead of re-entering these commands from the keyboard.

GAID is coded to occupy 1213 bytes of memory, leaving a large area for the user's program. (Additional space is also required for the patch area and zero sector pointer area specified by the user.) If the user requires additional core space when debugging, a subset of GAID called the Basic Debugging Aid (BAID) is provided (Section 21). BAID is about 450 bytes smaller than GAID, but provides a basic debugging capability.

GAID is designed to work with the minimum Varian 520/i configuration of a 4K memory and a Model 33TC Teletype. All commands except I and J (see Figure 21.1) will function with any size of memory. The program requires the Binary Loader in the top of core to execute the L command. No other software routines are required.

Basic GAID Commands

The basic commands used to control GAID are summarized in Figure 15.1 and described in detail below. The X, Y and Z symbols represent hexadecimal numbers.

DXXXX.

Display memory byte XXXX and the following byte.

DXXXX-YYYY.

Display memory from address XXXX to address YYYY.

The teletype lists each 16th address in the interval and lists the memory in an array with 16 columns.

AXXXX: YV.

Alter the contents of memory location XXXX to YV.

AXXXX: YV, . . . YY.

Alter consecutive memory bytes beginning with location XXXX until a period is entered after a YY entry. (The comma is optional.)

PXXXX-YYYY, ZZZZ.

Punch memory to paper tape from location XXXX to location YYYY.

Leader and trailer are output. Punch must be turned on manually after the period is entered during the leader output, and must be turned off at end of output.

ZZZZ is the transfer address. If other areas of core are to be added to the tape (by entering another punch command), ZZZZ should be negative (\$8000). If ZZZZ is negative, the computer halts after the tape is punched so the user

AXXXX-YY.	Alter memory byte XXXX.
AXXXX-YY,, YY.	Alter consecutive memory bytes (comma optional).
B.	Branch to P register location. Restore conditions.
CR ----.	Display register R.
CR ----, YYYY.	Change register R to YYYY.
DXXXX.	Display two bytes at location XXXX.
DXXXX-YYYY.	Display consecutive bytes from XXXX to YYYY.
EXXXX.	Erase an Insert or Dump patch at location XXXX.
FXXXX-YYYY.	Form patch area from XXXX to YYYY.
G.	Go to paper tape command input.
IXXXX-YY.	Insert byte YY before location XXXX.
IXXXX-YY, . . ., YY.	Insert bytes before location XXXX.
JXXXX, YYYY-ZZZZ.	Jump to dump bytes YYYY to ZZZZ before executing instruction at location XXXX.
K.	Keyboard command input mode.
L.	Load an object tape.
MXXXX.	Branch to subroutine at XXXX and return to GAID.
NXXXX-YYYY.	Name the zero sector pointer area from XXXX to YYYY.
PXXXX-YYYY, ZZZZ.	Punch object tape of memory locations XXXX to YYYY and punch transfer address ZZZZ.

Figure 21-1. Table of Commands for GAID

general debugging (GAID)

may turn off the punch then depress RUN. The next punch command causes the computer to halt so the user may turn on the punch, then depress RUN. This feature permits output of non-contiguous areas of core by preventing extraneous characters, such as line feed or carriage return, to be punched on the object tape, and by suppressing leader/trailer output between records.

CR----

CR---, YYYY.

Change register R.

The teletype lists contents of a register R (or precision S, environment E, or overflow F) as they are stored in the conditions table.

The user may type a period if only display is desired. To alter the stored contents, a comma and the new contents followed by a period are typed. The registers which can be displayed are A, B, C, D, P, Q, X, and Y.

Precision is identified by the letter S and is represented by a 4-character number 0001 to 0004.

Overflow is identified by the letter F. 0000 indicates overflow not set and 0001 indicates overflow set.

Environment is identified by the letter E. 0000 indicates that the computer is in the interruptable environment and 0001 indicates that the computer is in the non-interruptable environment. The computer is always in the environment specified by the "CE" stored value.

All of the above conditions (prior to being changed by the user) are those which have been stored when the user entered GAID through a branch (BRU \$ED35) or a branch and mark (BRM \$380D33). This allows user to observe all of these conditions as they were during any part of the execution of his program.

B.

Branch to the location specified in the P register storage.

This command causes all of the conditions described above (registers, environment, overflow, and precision) to be set up as they have been specified by the user (through CR command). Execution then begins at the location specified in the P register storage. If any of the conditions have not been specified by the user, they have the value which has previously been stored. The initial values of the conditions when GAID is loaded into the computer are zero for all registers, overflow and environment, and 2 for precision.

MXXXX.

Branch and Mark to user's subroutine at location XXXX, setting registers, environment, overflow, and precision as they are stored in the conditions table.

Upon completion of the subroutine, control returns to GAID, storing all conditions in the conditions table as they were at completion of the subroutine.

This command allows a user to set up a calling sequence for his subroutine using the CR command, execute the subroutine, then return to GAID and examine the results of execution by displaying the stored conditions by use of the CR command. The P register is not stored on return.

L .

Load object tape

The user should set the B register for desired mode before execution. Zero will cause the loader to halt after loading and non-zero will cause the loader to transfer after loading.

G .

Go to tape input.

The teletype turns on the paper tape reader and the program accepts commands from paper tape until command K is received from the paper tape. Rubout, carriage return, line feed and blank characters are ignored.

K .

Keyboard input.

The teletype terminates the paper tape reader and accepts further commands from the keyboard.

/ .

Cancel command.

If / is input during the entering of a command, GAID terminates the com-

mand and outputs a carriage return and line feed. Another command may then be input. If this is done during an Alter command, only the last byte input does not enter memory since each byte is stored as soon as it is entered. A similar exception occurs during patching commands (see below).

IXXXX: YY .

Insert Byte YY before memory location XXXX.

IXXXX: YY,YY,...YY .

Insert the consecutive bytes YY before memory location XXXX.

JXXXX: YYYY,ZZZZ .

Jump to routine which lists the contents of cells YYYY to ZZZZ each time the instruction at XXXX is executed.

The display occurs immediately preceding the execution of instruction XXXX. The command is implemented by inserting a patch which is the calling sequence for the display routine. The display routine saves and restores all conditions before and after execution.

FXXXX-YYY .

Form the temporary patch area beginning at location XXXX and ending at location YYYY.

The patch area may be located in a particular sector to accommodate "indirect-to-current" memory reference instructions inserted into the patch area.

general debugging (GAID)

If GAID is used after patches have been made which are no longer wanted, new patches can be overlaid by using the F command to initialize the patch area pointers.

NXXXX-YYYY.

Name pointer area.

Pointers begin at XXXX and end at YYYY. Both values must be less than \$3FF. If the pointer area is overrun, a "O?" followed by carriage return and line feed is output.

EXXXX .

Erase an Insert or Dump patch from the program at location XXXX.

The branch to the patch area is removed from the user's program and the previous bytes are replaced. The space used in the patch area is not recovered unless the F command is used.

Construction of Patch Areas

When an Insert or Jump (display) command is given to GAID, the following sequence of events takes place:

1. Two bytes are removed from the user's program at location XXXX and placed in the patch area at the beginning of the patch for use by the erase command. These bytes are not executed until after the inserted instructions are executed.
2. A branch to the patch location is inserted into the user's program at location XXXX. This branch points to the

next location after the two bytes removed from the user's program.

3. The instructions YY input by the user are placed consecutively in the patch area. (In the case of a Jump command, the display routine call is placed in the patch area.) The two bytes removed from the user's program are then placed in the patch area. Additional bytes may be required to be relocated from the user's program to the patch area to insure an executable sequence of instructions is present (if the insert was made at a one byte instruction followed by a two- or three-byte instruction, three or four bytes are placed in the patch area at this point).
4. A branch back to the user's program to the next instruction is placed in the patch area completing the patch.

When an Insert or Jump command is given which relocates a memory reference instruction of the direct-to-current or indirect-to-current type from a user's program to the patch area, the instruction is not executable if the patch area is not in the current sector. Due to this fact, all direct-to-current and indirect-to-current relocated instructions are changed to indirect-through-zero and double-indirect-through-zero instructions. This is done by constructing a pointer in the zero sector in the area specified by the user (N command).

The user is cautioned about the following limitations of the patch and jump routines:

1. When a patch is initialized at a one-byte instruction, the instruction following (one, two, or three bytes long) is also

replaced (moved to the patch area). Therefore, a patch can be made neither before nor after a one-byte instruction.

2. A patch cannot be made which causes an entry point (referenced by a branch elsewhere in the program) to be relocated into the patch area.
3. A patch cannot be made which relocates nonexecutable data.
4. If a patch is made following a skip instruction, the entire patch is skipped when the skip condition is true.
5. If a patch is made at a skip instruction followed by two one-byte instructions or at a one-byte instruction followed by a skip instruction, execution fails when the skip condition is true. To avoid this condition, the user should alter the skip or the one-byte instruction followed by the skip to a NOP, then input the Insert command, with the bytes which were changed to NOP's entered as the last bytes of the insert.

If the patch area which has been designated by the F command is overrun during an Insert or Jump Display command, GAID outputs P? followed by a carriage return and line feed. If this occurs, the patch is not completed. If the pointer area is overrun, GAID outputs O? followed by a carriage return and line feed, and again the patch is not completed. To correct these errors, the user must first erase the patch which caused the overrun, then designate a new patch or pointer area, then repeat the original command.

The operation of the Insert and Jump to display commands above 4K memory will cause erroneous results in most cases.

Special Displays

It is possible to display the contents of stored registers and conditions through the J command by designating the desired area of the conditions storage table. The addresses of the various conditions stored in the table are as follows:

Address	Condition
\$0D7F	A Register
\$0D81	B Register
\$0D83	C Register
\$0D85	D Register
\$0D87	Environment
\$0D89	Overflow
\$0D8B	P Register
\$0D8D	Q Register
\$0D9B	X Register
\$0D9D	Y Register

For example, if it is desired to display the contents of the X register immediately preceding the execution of the instruction at XXXX, the user would enter the command "JXXXX, 0D9B-0D9C. (Each condition is a two-byte quantity.)

Another desirable operation is the display of the patch area and pointer-area pointers. Knowledge of these values also allows the

general debugging (GAID)

user to save these areas for later use. The locations of these pointers is as follows:

Location	Label	Function
\$0B6E	OP	Next available pointer area location
\$0B70	OE	Last available pointer area location
\$0B72	PN	Next available patch area location
\$0B74	PE	Last available patch area location

For example, if a user wants to punch an object tape of a patched program, he can display \$0B6E to see the amount of patch area used, and then punch the proper area.

Overlaying

It is possible to overlay a portion of GAID and still execute the remaining portions. The overlaying can be done in steps as follows (see Figure 21.2).

Overlaying between \$AE3 and \$BOE deletes the J command

Overlaying between \$BOF and \$C59 deletes E, F, I, and N commands

Overlaying between \$C5A and \$C6B deletes the L command

Overlaying between \$C6C and \$D1E deletes the P command

Overlaying between \$D1F and \$D32 deletes the M command

Overlaying between \$D33 and \$D7E deletes the BRU or BRM to GAID functions

Any further overlaying will prevent the operation of the program entirely.

It is recommended that if the user expects to overlay any of the GAID, the BAID program should be used in its place. The BAID provides the same capability remaining after the second overlay above, but it is smaller than GAID, routine for routine, since it does not support the additional functions and constants needed for GAID. The BAID program can be overlaid to location \$E29 before operation is prevented.

Operation of GAID

GAID is loaded into the computer using the Binary Loader (Section 21). Upon being loaded, the routine outputs a carriage return and line feed and then enters the keyboard I/O mode, awaiting a typed-in command.

To manually enter GAID, the user sets the P register = \$EOC and depresses RUN. To enter GAID from a user's program, a branch to debug instruction (\$ED35) is inserted at the desired branch location. To save the location from which GAID was entered, the user may insert a branch and mark to debug in his program (\$380D33).

The user is cautioned in the keyboard mode, against depressing the teletype keys too rapidly. If the keys are struck at a rate

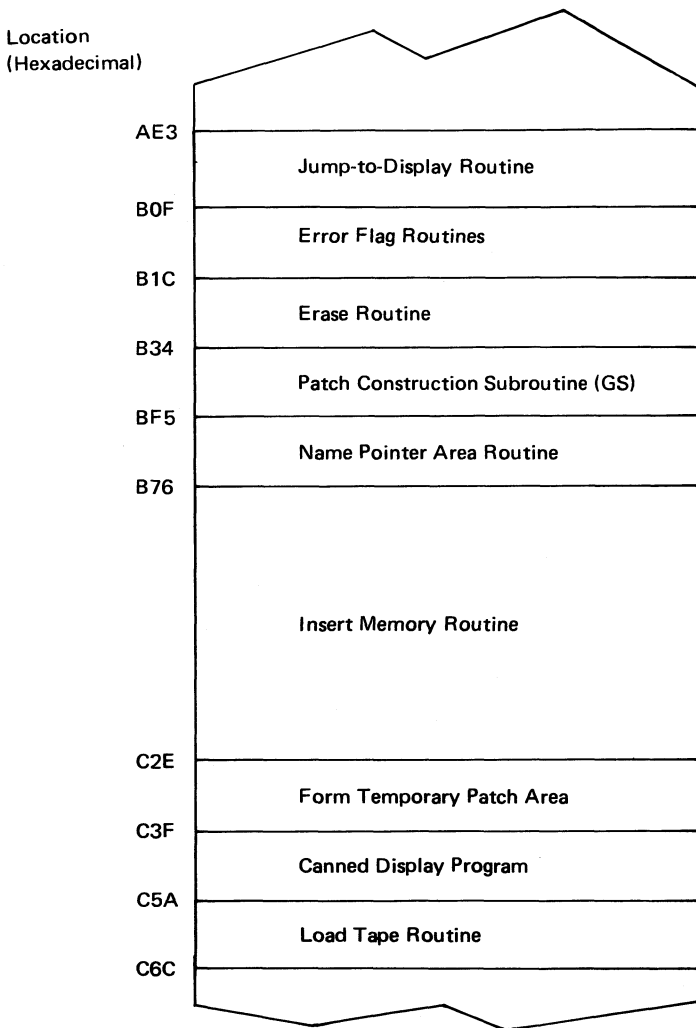


Figure 21-2. Varian 520/i General Debugging Aid (GAID) Memory Map (Sheet 1)

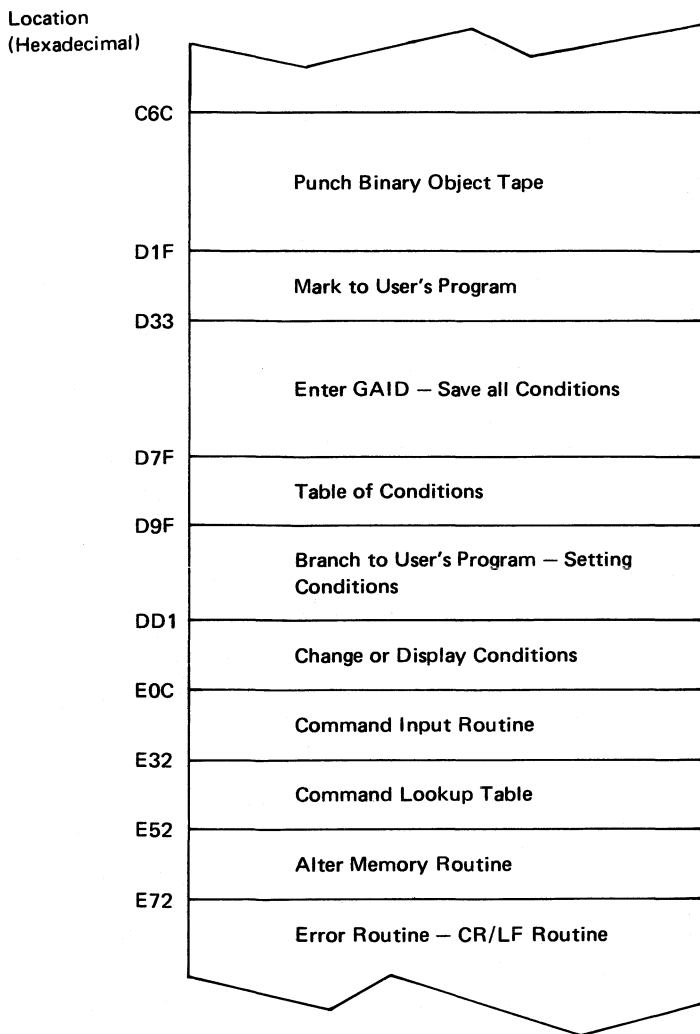


Figure 21-2. Varian 520/i General Debugging Aid (GAID) Memory Map (Sheet 2)

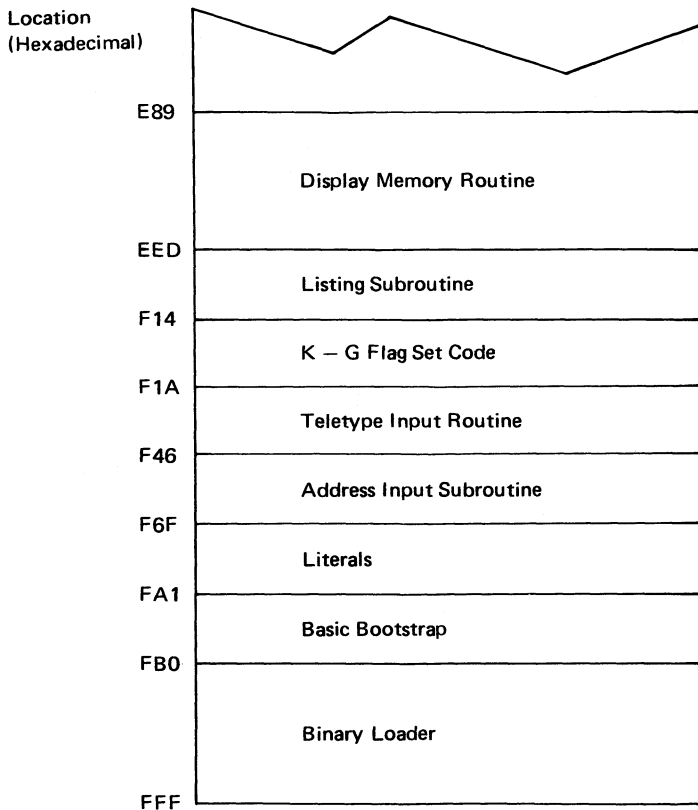


Figure 21-2. Varian 520/i General Debugging Aid (GAID) Memory Map (Sheet 3)

general debugging (GAID)

greater than five characters per second, improper data will be entered into GAID and the results are unpredictable.

All commands must be entered in the specified format. For example, all addresses

must be four characters, and punctuation must be entered correctly.

If an error is detected by GAID such as an incorrect command, or incorrect format, a question mark, followed by a carriage return and line feed, is output. Another command may then be typed.

SECTION 22 — BASIC DEBUGGING AID (BAID)

Identification No. 92A0203-0036

Features:

- Subset of General Debugging Aid (GAID) for use when user's program occupies most of core
- Occupies 430 bytes of core less than the General Debugging Aid
- Allows partial overlay of functions by user's program without stopping basic operation of displaying and altering core

The Basic Debugging Aid (BAID) is a system of routines which provides on-line debugging capability to the user. BAID is a subset of the General Debugging Aid (GAID) and is provided for use when the program to be debugged is too large to allow space for GAID. BAID provides all the necessary features for debugging, but omits the complex features provided by GAID for automatic patching. BAID allows the user to perform the following functions:

1. Display and alter variable fields of memory.
2. Branch to a program with all conditions of the system specified.
3. Branch from a program to BAID and save all conditions at the time of the branch.

4. Produce a loadable punched paper object tape of desired areas of core.

One feature of BAID is the ability to command it from paper tape as well as from keyboard. This enables a user to prepare a program debugging sequence off-line before the computer is available. The user may then load his program and ready his tape of commands for BAID when the computer becomes available. After execution, he may analyze the results of his efforts (again off-line) by reading the teletype listing produced. If additional debugging is required, the user may enter previous command tapes through the tape reader to set up his program changes instead of re-entering these commands from the keyboard.

BAID is coded to occupy only 783 bytes of memory, leaving a maximum amount for the user's program. In addition, the organization of BAID is designed to allow a user to partially overlay it with his program and still have available the basic features needed for debugging. This is done in steps so that as an increasing amount of BAID is overlaid, an increasing number of functions are lost, but the basic debugging capabilities are saved.

BAID is designed to work with the minimum Varian 520/i configuration of a 4K memory and a Model 33TC Teletype. No

basic debugging aid (BAID)

sense switch settings or other software routines are required to operate BAID.

Basic BAID Commands

The basic commands used to control BAID are summarized in Figure 22.1 and described in detail below. The X, Y, and Z symbols represent hexadecimal numbers.

DXXXX.

Display memory byte XXXX and the following byte.

DXXXX-YYYY.

Display memory from address XXXX to address YYYY.

The teletype lists each 16th address in the interval and lists the memory in an array with 16 columns.

AXXXX:YY.

Alter the contents of memory location XXXX to YY.

AXXXX:YY,YY....YY.

Alter consecutive memory bytes beginning with location XXXX until a period is entered after a YY entry. (The comma is optional.)

PXXXX-YYYY, ZZZZ.

Punch memory to paper tape from location XXXX to location YYYY.

Leader and trailer are output. Punch must be turned on manually after the period is

entered during the leader output, and must be turned off at end of output.

ZZZZ is the transfer address used by the loader. If other areas of core are to be added to the tape (by entering another punch command), ZZZZ should be negative (\$8000). If ZZZZ is negative, the computer will halt after the tape is punched so the user may turn off the punch then depress RUN. The following punch command will cause the computer to halt so the user may turn on the punch then depress RUN. This feature prevents extraneous characters such as line feed and carriage return to be punched on the object tape, and suppresses leader/trailer output between records.

CR----

CR---, YYYY.

Change register R.

The teletype lists the contents of a register R (or precision S, environment E, or overflow F) as they are stored in the conditions table.

The user may type a period if only display was desired. To alter the stored contents, a comma and the new contents followed by a period are typed.

The registers which can be displayed are A, B, C, D, P, Q, X, and Y.

Precision is identified by the letter S, and is represented by a four character number 0001 to 0004.

Overflow is identified by the letter F. 0000 indicates overflow not set, and 0001 indicates overflow set.

AXXXX-YY	Alter memory byte XXXX.
AXXXX-YY, . . . , YY.	Alter consecutive memory bytes (comma optional).
B.	Branch to P register location. Restore conditions.
CR ----	Display register R.
CR ----, YYYY.	Change register R to YYYY.
DXXXX.	Display two bytes at location XXXX.
DXXXX-YYYY.	Display consecutive bytes from XXXX to YYYY.
G	Go to paper tape command input.
K	Keyboard command input mode.
MXXXX.	Branch to subroutine at XXXX and return to GAID.
PXXXX-YYYY,ZZZZ.	Punch object tape of memory from location XXXX to YYYY and punch transfer address ZZZZ.

Figure 22.1 Table of Commands for GAID

Environment is identified by the letter E. 0000 indicates that the computer is in the interruptable environment, and 0001 indicates that the computer is in the non-interruptable environment. The computer is always in the environment specified by the "CE" stored value.

All of the above conditions (prior to being changed by the user) are those which have been stored when the user entered BAID through a branch (\$ED5A) or a branch and mark (\$380D58) from his program. This allows the user to observe all of these conditions as they were during any part of the execution of his program. (BRM to BAID saves the "branched from" location in

the P register storage, but BRU to BAID does not save the P register.)

B.

Branch to the location specified in the P register storage.

This command causes all of the conditions described above (registers, environment, overflow, and precision) to be set up as they have been specified by the user (through the CR command). Execution then begins at the location specified in the P register storage. If any of the conditions have not been specified by the user, they have the value which has previously been stored. The initial values

basic debugging aid (BAID)

of the conditions when BAID is loaded into the computer are zero for all registers, overflow and environment, and two for precision.

MXXXX.

Branch and Mark to user's subroutine at location XXXX setting registers, environment, overflow, and precision as they are stored in the conditions table.

Upon completion of the subroutine, control returns to BAID storing all conditions in the conditions table as they were at completion of the subroutine.

This command allows a user to set up a calling sequence for his subroutine using the CR command, execute the subroutine, then return to BAID and examine the results of execution by displaying the stored conditions by use of the CR command. The P register is not stored on return.

G.

Go to tape input.

The teletype turns on the paper tape reader and then accepts commands from paper tape until command K is received from paper tape. Rubout, carriage return, line feed, and blank characters are ignored.

K.

Keyboard input.

The teletype terminates the paper tape reader and accepts further commands from the keyboard.

/

Cancel command.

If / is input during the entering of a command, BAID terminates the command and outputs a carriage return and line feed. Another command may then be input. If this is done during an Alter command, only the last byte does not enter memory since each byte is stored as soon as it is entered.

Overlaying BAID

The ability to overlay a portion of BAID and still execute the remaining portions is illustrated by the memory map in Figure 22.2.

Overlaying can be done in steps as follows:

Overlaying between \$C91 and \$D43 deletes the P command

Overlaying between \$D44 and \$D57 deletes the M command

Overlaying between \$D58 and \$D9C deletes the BRU or BRM to BAID functions

Overlaying between \$D9D and \$E29 deletes the B and C commands

Any further overlaying will prevent the operation of the program entirely.

Operation of BAID

BAID is loaded into the computer using the Binary Loader (Section 22). Upon being loaded, the routine outputs a carriage return and line feed and then enters the keyboard I/O mode, awaiting a typed-in command. To manually enter BAID, the user sets the P

Hex Location

C91

Punch Binary Object Tape

D44

Mark to User's Program

D58

Enter BAID – Save all Conditions

D9D

Table of Conditions

DBD

Branch to User's Program – Setting Conditions

DEF

Change or Display Conditions

E2A

Command Input Routine

E50

Command Lookup Table

E70

Alter Memory Routine

E90

Error Routine – CR/LF Routine

EA7

Display Memory Routine

F04

Listing Subroutine

F2B

K-G Flag Set Code

F33

Teletype Input Routine

F5F

Address Input Subroutine

F88

Literals

FA1

Basic Bootstrap

Binary Loader

Figure 22.2 Basic Debugging Aid (BAID) Memory Map

basic debugging aid (BAID)

register to \$E2A and depresses RUN. To enter BAID from a user's program, a branch to debug instruction (\$ED5A) is inserted at the desired branch location. To save the location from which BAID was entered, the user may insert a branch and mark to debug in his program (\$380D58).

The user is cautioned against depressing the teletype keys too rapidly. If the keys are struck at a rate greater than five characters per second, improper data will be entered into BAID and the results are unpredictable.

All commands must be entered in the specified format. For example, all addresses must be four characters, and punctuation must be entered correctly.

If an error is detected by BAID such as an incorrect command, or incorrect format, a

question mark, followed by a carriage return and line feed is output. Another command may then be typed.

The Binary Loader may be called from BAID by setting the proper values in the registers, then using the B command. For example, if the Loader is located in the third sector (\$C00-\$FFF), the following sequence of commands is used:

```
CP   XXXX, OFBD.  
CQ   XXXX, OFF4.  
CS   XXXX, 0001.  
CB   XXXX, 0000. (or non-zero  
           if desired)  
(Tape is readied in tape reader)  
B. (Loader begins execution)
```

SECTION 23 – BASIC COMPUTER TEST PROGRAM

Identification No. 92A0103-005A

Features:

- Tests all non-I/O instructions in all address modes for all precisions in both environments
- Tests for proper operation of the teletype keyboard, page printer, paper tape reader, and punch
- Tests all of the available memory for unique addressing and worst case pattern noise immunity
- All of the tests may be entered through keyboard control, along with controlling parameters
- Selection of messages print out or program halt upon the detection of an error
- The entire test program resides within the basic 4K computer and is loadable by one object tape

The Varian 520/i Basic Computer Test Program is designed to enable the user to determine whether the basic Varian 520/i computer configuration is functioning properly, or, if not, to give aid in determining the nature of the malfunction.

Through the use of the teletype keyboard and the sense switches, the user has full

control over the execution of the Test Program.

The program consists of the following major sections:

Executive Routine

Instruction Tests

Teletype Tests

Memory Tests

The discrete tests are each addressable by typing a corresponding four letter mnemonic (see Figure 23-1), and may be run once or continuously through sense-switch selection.

The consistent assignment of sense-switch definitions has been maintained throughout all of the tests to provide a uniform operating interface.

The user may select, through a sense-switch setting, a mode which causes the program to halt upon the detection of each error condition, allowing that error to be explored in detail. Or, as an alternative, the test program will continue to completion, producing a standardized error printout upon the detection of each error.

basic computer test program

The sense switches have in general the following assigned meaning for all tests:

<u>Sense Switch</u>	<u>Function Set/Reset</u>
1	Halt on error/type error and continue test
2	Continuously repeat test/return to Executive on completion of test
3	Immediately return to Executive/proceed to completion of test

The Basic Computer Test Program operates with a minimum Varian 520/i configuration consisting of 4K memory with a Model 33TC ASR teletype.

All input/output communication to the test programs is accomplished through the teletype. Input requests to execute tests are accepted through the keyboard and error messages are printed on the teletypewriter.

The complete set of basic tests are contained on a single object tape approximately 20 feet in length. The loading of this tape requires approximately two minutes and readies the computer for executing all of the contained tests.

Each separate test is constructed as a closed subroutine and is called by the Executive Routine upon execution. By adding additional test mnemonics and precanned parameters to the Executive Routine call table, additional tests may be added to the package.

In addition, each of the operand tables for the individual instruction tests are constructed so that the number and variety of

entries may be altered. The termination of each table is determined by an EQU statement that is automatically repositioned when adding additional data statements.

Executive Routine

The Executive Routine provides control and communications to and from each of the various test sections. It accepts the identification name along with any desired input parameters that may be specified and calls the requested test. It monitors the sense switches and establishes the requested modes.

The keyboard input format for requesting a test is as follows:

ABCD

or

ABCD,XXXXXXXX

Where ABCD is the four letter mnemonic defining the test desired (Figure 23-1) and X...X is the specified input parameters.

If the user elects not to type in specific parameters for the test, the Executive Routine selects parameters stored in a table which is part of the program.

These preselected parameters have been chosen to produce the most general test insofar as possible. The detailed description of each of the tests defines these parameters.

The typed in parameters consist of eight consecutive hexadecimal characters and occupy the 32-bit accumulator upon entrance to the specified test.

Title	Mnemonic
Instruction Test	INST
One Byte NMR Test, Part A	INSA
One Byte NMR Test, Part B	INSB
Two Byte NMR Test	INSC
Two Byte MR Test	INSD
Keyboard Input/Teletypewriter Output Test	TTYA
Paper Tape Reader/Punch Test	TTYB
Teletypewriter Rotating Pattern Test	TTYC
Memory Test	MEMO

Figure 23-1. Mnemonics for Individual Tests

basic computer test program

The test is entered upon receipt of the period indicating completion of the request. When the test is completed, the Executive Routine signals the user by repeating the type out of the test mnemonic.

If an error in the typed in request is detected, the routine responds with a question mark followed by a carriage return and line feed. The user may then retype the request.

Instruction Tests

The Instruction Tests have been designed to test all of the 520/i instructions in a reasonably thorough manner. As applicable, all precisions, all addressing modes and both environments are applied to all of the instructions for many operands. Upon detection of an error condition, the program either halts or types out an error message in the following format (as determined by sense switch option):

PPPP AAAA/CCCC XXXX

where:

PPPP	is the location of the test sequence that detected the error
AAAA	is the 16 MSB of the accumulator
CCCC	is the 16 LSB of the accumulator
XXXX	is the contents of the index register

The registers displayed will be for the environment in which the error condition was detected.

If the program is instructed to halt for an error condition, further useful information may be obtained by displaying certain registers and memory locations. The detailed procedure for this is defined under each test description. The various instructions have been grouped into several categories (for testing purposes) and each of these tests are described as follows:

Instruction Test (INST)

This consists of all of the individual instruction tests executed sequentially. The input format to the Executive Routine to request this test is:

INST.

One Byte Non-memory Reference Instruction Test, Part A (INSA)

This instruction test checks a sub-class of the one byte non-memory reference instructions. These instructions are checked in all precisions, both environments and for six different operands. The input format to the Executive Routine to request this test is:

INSA.

If an error condition is detected in the "type error" mode, the instruction(s) in error are as shown in Figure 23-2.

If an error condition is detected in the "halt on error" mode, the following procedure may be performed to obtain further information concerning the error:

If the P register displays \$0281*, then display the contents of JR and JR + 1

Message (Hexadecimal)	Test	Error Condition
0109	INST	Change of environment (interruptable to noninterruptable)
0110	INST	Skip on interruptable environment (no skip condition)
0116	INST	Change of environment (noninterruptable to interruptable)
011E	INST	Skip on interruptable environment (skip condition)
0123	INST	CLA, SAZ (skip condition)
012B	INST	SPI, IXO (preliminary test)
012F	INST	BRU (preliminary test)
013A	INST	LOD, STO, SUB (preliminary test)
0141	INST	SPI, BRUX (preliminary test)
0147	INST	SP2, BRUX (preliminary test)
014D	INST	SP3, BRUX (preliminary test)
0153	INST	SP4, BRUX (preliminary test)
015B	INST	IAR, SOV (preliminary test)

Figure 23-2. Error Messages (Sheet 1)

basic computer test program

Message (Hexadecimal)	Test	Error Condition
0197	INSA	SAZ, A = zero
01A0	INSA	SAZ, A = non-zero
01AA	INSA	SAZN, A = non-zero
01B3	INSA	SAZN, A = zero
01BB	INSA	SOD, A = odd
01C2	INSA	SOD, A = even
01CA	INSA	SODN, A = even
01D1	INSA	SODN, A = odd
01D9	INSA	SAN, A = negative
01E0	INSA	SAN, A = positive
01E8	INSA	SANN, A = positive
01EF	INSA	SANN, A = negative
01F8	INSA	SRI
0201	INSA	SRI/SLI, A = positive
0208	INSA	SRI/SLI, A = negative
0211	INSA	RRI, A = even
0218	INSA	RRI, A = odd
0221	INSA	RLI, A = odd

Figure 23-2. Error Messages (Sheet 2)

Message (Hexadecimal)	Test	Error Condition
0228	INSA	RLI, A = even
0231	INSA	SOV, overflow = reset
0237	INSA	SOV, overflow = set
023E	INSA	SOVN, overflow = set
0246	INSA	SOVN, overflow = reset
024D	INSA	CLA
02FD	INSB	1 Byte NMR instruction sequence
04FB	INSC	2 Byte NMR instruction sequence
0579	INSD	Direct mode address error
058A	INSD	Direct/Current Mode address error
059B	INSD	Indirect/Current Mode address error
05B6	INSD	Index Mode address error
05C7	INSD	Indirect/Zero Mode address error
083B	MEMO	Unique address by sector error
086D	MEMO	Unique address by byte error
08FC	MEMO	Worst case pattern error

Figure 23-2. Error Messages (Sheet 3)

(\$027B, \$027C). If the P register displays \$028E, then display the contents of JS and JS + 1 (\$0288, \$0289). This

corresponds to the error addresses as shown in Figure 23-2. Referencing this table yields the instruction(s) in error.

Display the contents of JR (\$0297). This gives the precision in which the error occurred, as follows:

\$10 corresponds to precision 1

\$11 corresponds to precision 2

\$12 corresponds to precision 3

\$13 corresponds to precision 4

Display the contents of the X register. This contains the address of the data being used at the occurrence of the error.

Display the contents of JX (\$129A). This gives the environment in which the error occurred, as follows:

\$0000 Corresponds to the interruptible environment

\$0001 Corresponds to the non-interruptible environment

One Byte Non-memory Reference Instruction Test, Part B (INSB)

This instruction test checks the remaining one byte non-memory reference instructions not checked by Part A. These instructions are checked in all precisions, both environments and for three different operands. The input format to the Executive Routine to request this test is:

INSB.

If an error condition is detected in the "type error" sense switch mode, the instructions in error are as shown in Figure 23-2. If an error condition is detected in the "halt on error" mode, the following procedure

may be performed to obtain further information concerning the error.

Display the contents of JU (\$0297). This gives the precision in which the error occurred, as follows:

\$10 corresponds to precision 1

\$11 corresponds to precision 2

\$12 corresponds to precision 3

\$13 corresponds to precision 4

Display the contents of NU (\$0322-\$0325). These sequential bytes represent (to the encountered precision) the data being operated on.

Display the contents of NR (\$0356). This gives the environment in which the error occurred, as follows:

\$0000 corresponds to the interruptible environment

\$0001 corresponds to the non-interruptible environment

To determine the unique instruction failure, sequence through the instruction test starting at ND (\$02E4), examining the effects of executing each instruction.

Two Byte Non-memory Reference Instruction Test (INSC)

This instruction test checks the two-byte non-memory reference instructions in both environments for four data patterns.

Every combination of every register operation code, with every register as source and every register as destination, is checked with

various data patterns with the following exceptions:

1. The class of instructions for which the pseudo zero register occurs as a destination register is not checked.
2. The class of instructions for which the P register occurs as a destination register is not checked.
3. The class of instructions for which the P register occurs as a source register is checked for only one data value.

The input format to the Executive Routine to request this test is:

INSC.

If an error condition is detected in the "type error" mode, the instructions in error are as shown in Figure 23-2. If an error condition is detected in the "halt on error" mode, the following procedure may be done to obtain further information concerning the error.

Display the contents of KD 1 (\$0462). This will give the source register and the destination register being executed at the time of failure as follows:

\$SD

where:

S corresponds to the source register and D corresponds to the destination register. (See Figure 23-3 for the numeric to alphanumeric correspondence.)

Display the contents of KU (\$0515). This gives the environment in which the error occurred, as follows:

\$0000 corresponds to the interruptable environment

\$0001 corresponds to the non-interruptable environment

Display the contents of KY, $KY \pm 1$ (\$0404, \$0405). This gives the data packet address that contains the data being operated on.

To determine the unique instruction failure, restore the data word into the A register and sequence through the instruction test starting at KC-2 (\$045D), examining the effects of executing each register transfer command.

Two Byte Memory Reference Instruction Test (INSD)

This instruction test checks the two byte memory reference instructions in all precisions, in both environments, for six operands and for the following address modes:

Direct Mode

Direct/Current Mode

Indirect/Current Mode

Index Mode

Indirect/Zero Mode

The input format to the Executive Routine to request this test is:

INSD.

[illegible]

Figure 23-3. Hexadecimal Representation of all Acceptable ASCII Characters

If an error condition is detected in the "type error" mode, the instructions in error are as shown in Figure 23-2.

If an error condition is detected in the "halt on error" mode, the following procedure may be done to obtain further information concerning the error.

Display the contents of JU (\$0297). This gives the precision in which the error occurred, as follows:

- \$10 corresponds to precision 1
- \$11 corresponds to precision 2
- \$12 corresponds to precision 3
- \$13 corresponds to precision 4

Display the contents of the X register. This contains the address of the data being used at the occurrence of the error.

Display the contents of LR (\$05EF). This gives the environment in which the error occurred, as follows:

- \$0000 corresponds to the interruptable environment
- \$0001 corresponds to the non-interruptable environment

Restore the operand data in the accumulator and sequence through the instruction test for the applicable address mode, examining the effects of executing each instruction.

Teletype Tests

The Teletype Test sub-programs provide the user with the capability of testing all func-

tions of the ASR Model 33TC Teletype, including keyboard input, teletypewriter output, and paper tape input and output. Because of the nature of the three tests, Sense Switch 1 (Halt on error/Type error and continue test) is not used.

Keyboard Input/Teletypewriter Output Test (TTYA)

This test exercises the teletype keyboard input and the teletypewriter output. A character that has been typed is input, verified in the acceptable character table, then output to the teletypewriter. The teletype is then conditioned to receive another typed character.

To verify the non-printing characters, the paper tape punch may be turned on and the input and output punched characters compared.

A halt occurs upon the receipt of an illegal character, with the erroneous bit pattern in the A register. Depressing RUN re-initializes the test. Control is returned to the Executive Routine by setting Sense Switch 3 and typing any additional character.

The input format to the Executive Routine to request this test is:

TTYA.

It is suggested that in order to thoroughly exercise this test all possible keyboard inputs be tried as shown in Figure 23-3.

Paper Tape Reader/Punch Test (TTYB)

This test exercises the teletype paper tape reader and punch. A block of data as

basic computer test program

specified by the input parameters is punched out on paper tape. A halt then occurs, allowing the beginning of the punched tape to be placed in the paper tape reader. If the specified block causes the paper tape to be too short to be read in the reader, sufficient repetitive blocks are punched prior to the halt. Depressing RUN then causes the test to alternately punch a block and read a block in a continuous manner, checking for any errors.

On the occurrence of an error, a halt occurs with the erroneous bit pattern in the A register. The paper tape may be compared to determine if the cause was the punch or the reader.

The input format to the Executive Routine to request this test is:

TTYB.
or TTYB,XXYY0000.

where:

XX corresponds to the first value of the ascending binary data sequence and YY corresponds to the final value of the ascending binary data sequence

The detailed operating procedure is as follows:

Type in the test request message with or without parameters as desired. If no parameters are typed, the punched and read data consists of all ascending binary data from \$00 to \$FF.

The test program halts upon completion of the input request. The punch must be turned on and RUN depressed.

The requested data is punched out and the test program again halts. If the requested data does not produce sufficient length of tape to reach the reader, the data pattern is repeated until it does. Ready the first desired character in the paper tape reader. For the assumed parameters this is \$00 prior to \$01.

Depressing RUN now causes the test to continuously punch a block and read a block. If an error is detected a halt occurs at \$0733. The A register contains the character as read, and the C register contains the data bits in error.

Momentarily setting Sense Switch 3 causes the test program to return to the Executive Routine.

Teletypewriter Rotating Pattern Test (TTYC)

This test exercises the teletypewriter by outputting every printable character for one 64 character line and then advancing the position of the characters of succeeding lines for 64 lines. This results in every printable character to appear in every print position (for the first 64 positions).

The input format to the Executive Routine to request this test is:

TTYC.

The test proceeds to type the pattern, upon being requested, and runs to completion, returning control back to the Executive Routine. If it is desired to prematurely abort the test, momentarily setting Sense Switch 3 causes the test to immediately return to the Executive Routine.

Because of the symmetric nature of the printed pattern, it may be readily examined for the presence of any errors.

Memory Test

The Memory Test consists of a unique address sub-test and a worst case pattern sub-test.

The input format to the Executive Routine to request this is:

MEMO.
or MEMO,XXXXXXXX.

where:

XXXX corresponds to the low order address of the tested block and YYYY corresponds to the worst case bit pattern

If MEMO. is typed in, the first 4096 bytes of memory is exercised by the unique address sub-test, followed by the worst case pattern test. Successive running causes these two tests to be alternately executed. The assumed worst case pattern is bits 1 and 7 set.

If a 4096-byte block of memory other than the first one is to be tested, this is accomplished by inputting the lowest block address of that block as the XXXX parameter. If a worst case pattern other than the 1,7 bit pattern (\$0082) is desired, then the new pattern of bits is entered as the YYYY parameter.

If an error condition is detected in the "type error" mode, the instructions in error are as shown in Figure 23-2. If an error condition is detected in the "halt on error"

mode, the following procedure may be done to obtain further information concerning the error.

Display the contents of the P register. This will give the type of memory error as shown in Figure 23-2.

Display the contents of MW, MW ± 1 (\$09B3, \$09B4). This will give the address of the memory word in error.

The unique address test is first performed by determining that there are 16 unique 256 byte sectors per 4096 byte block. Then each 256 byte sector is determined to have 256 unique bytes. This permits all 4096 bytes to be uniquely identified even though only an eight bit memory address exists.

The test for worst case pattern consists of writing and reading a pattern (and its complement) in memory which is a function of the manner in which the sense winding is wound. Since the test is an individual bit test, all bits can either be 1's or 0's to simplify implementing the test.

The test requires that the memory be addressed sequentially from the lowest address of the block. Start data can be either 1's or 0's. The start data is written into memory sequentially and is complemented according to the following algorithm:

$$f = A_n \oplus A_m \oplus A_p \oplus A_g \oplus \dots$$

where:

A_n is the least significant address bit

A_m the next, etc.

$\oplus$ is the inequivalence symbol (exclusive or)

basic computer test program

In executing the Memory Test in which the memory block to be tested is occupied by the Memory Test, the lower half of the 4096-byte block is first tested, then the Memory Test is moved to the lower half. The upper half of the block is then tested followed by the Memory Test being restored in the upper half. As a result, all data and progress contained in the upper half (such as

the Basic Bootstrap, the Binary Loader, Basic Aid, etc.) are maintained through the testing of the entire block. However, the Instruction Tests and Teletype Tests which occupy the lower half are not maintained during the execution of the Memory Test. Therefore, if all of the tests are to be sequentially executed, the Memory Test must be executed last.

SECTION 24 – PERIPHERAL OPERATING SYSTEM (POSY)

Features:

- Various types of data may be exchanged among different peripheral devices with optional error checking taking place
- Because all peripheral tests are prepared in this high level language system, the user interface (input parameter requirements and error message formats) is standardized and consistent for all programs
- An in-depth understanding of the functioning of the test programs may be quickly gained because of the readability of the high level language
- Within the system, different peripherals may be referenced and various forms of data exchanged among them. Various transformations of data may be specified, including Hollerith to ASCII

The Peripheral Operating System (POSY) presents the user with a test-oriented, high-level language in which problem-directed test programs may be easily composed, thus freeing the user from having to make use of general test programs of a compromising nature.

The POSY program consists of a high level language compiler which produces an internally generated object program and pro-

ceeds to execute the generated object program to effect the desired test program. Various types of data may be caused to be exchanged between different peripheral devices with optional error checking taking place. Nine commands are available to the user to compose specific test programs.

The POSY system can operate within the minimum Varian 520/i configuration of 4K memory. However, any additional available memory may be used to extend the capability of a greater number of active peripherals and more complex test programs to be generated.

All test programs are prepared in a high level language and are entered via either the teletype keyboard or the teletype paper tape reader. Statements in this language consist of one of nine possible commands, followed by various possible operand fields. The entire format of the statement is free, each field need only to be separated by one or more spaces. Each statement must be terminated with a period.

Any statements terminated by a back slash (\) are ignored by the POSY compiler. The compiler also ignores carriage-return and line-feed characters.

The compiler indicates acceptable statements with a preceding asterisk. Unacceptable statements are not starred and an explanatory error message is generated.

peripheral operating system (POSY)

The POSY language recognizes three types of constants; octal, decimal and hexadecimal. These constants are self-defining or self-distinguishing according to the way in which they are represented. A decimal constant is represented by a string of decimal digits (0,1,2,3,4,5,6,7,8,9), the first of which is not zero. An octal constant is represented by a string of octal digits (0,1,2,3,4,5,6,7), the first of which must be zero. Hexadecimal constants are represented by a string of hexadecimal digits (0,1,2,3,4,5,6,7, 8,9,A,B,C,D,E,F), preceded by the character \$.

The nine commands that make up the POSY language are listed in Figure 24-1 and their structures are defined in the following paragraphs.

ASSIGN name mode device interrupt-line.

DATAi type parameter1 parameter2

CONTROL command.

INPUT name DATAi.

OUTPUT name DATAi.

CHECK DATAi.

LOOP n m.

PAUSE n.

BEGIN.

Figure 24-1. Summary of POSY
Operation Codes

ASSIGN name mode device interrupt line.

The execution of this statement causes the named peripheral for the given mode to be assigned the stated device number.

This permits all further references to point to that peripheral by name. The name field must be one of the symbolic names as defined in Figure 24-2. The mode must be one of the defined modes as defined in Figure 24-3. This, for example, permits distinction to be made between a peripheral connected on the 8-bit I/O bus and the same type of peripheral connected to the peripheral adapter or DMA port. The device field contains the physical device address of the peripheral as assigned in that particular installation. This latter number may be represented as a decimal, octal or hexadecimal value.

The interrupt-line field contains the interrupt line to which the peripheral is connected as defined in Table H. If the peripheral has been assigned to sense mode testing, the interrupt-line field must contain an "X".

An example of this statement might be:

ASSIGN CRDR A 5 X

This causes device address 5 to be assigned to the card reader which is to be tested under sense mode.

DATAi type parameter 1 parameter 2

The presence of this statement causes data generating subroutines with corresponding parameters to be made available to specify, control and generate various required data strings.

Symbolic Name	Peripheral
CRDR	SOROBAN Card Reader
HSPT	Remex/Tally High Speed Paper Tape Reader/Punch System
LINP	Data Products Line Printer
TTYK	ASR 33 TC Teletype Keyboard/Page Printer
TTYP	ASR 33 TC Teletype Paper Tape Reader/Punch

Figure 24-2. POSY Symbolic Names for Available Peripherals

The type field consists of a letter which corresponds to one of the several available types of data generators. Figure 24-4 gives the symbolic letters and the associated data patterns. The nature of the parameter field is dependent upon the type of data generator specified. This relationship is also shown in Figure 24-4. The "i" in the DATAi

operation is used to designate up to 10 different DATAi statements within a given test program and to provide the means for the other statements to reference them uniquely. The "i" may range from 0 to 9.

An example of this statement might be:

DATA3 A 1 100 0 100.

This causes an ascending binary sequence to be generated starting with the value 1 and terminating with the value 100.

INPUT name DATAi.

The execution of this statement causes the named peripheral to input characters (or words as applicable) of data to memory as specified by the DATAi field.

The name field is as defined under the ASSIGN statement. The DATAi field reference the earlier corresponding DATAi statement which defines the nature of the

Symbolic Character	Definition
A	8-bit input/output bus, sense mode
B	16-bit peripheral adapter
C	Direct memory access port
D	8-bit input/output bus, interrupt mode

Figure 24-3. POSY Symbolic Characters for I/O Modes

Symbolic Character/ Parameters	Definitions
DATAi A N M I L.	Data generator "A" generates a binary sequence of length L composed of concatenated ascending binary sequences starting with value N and terminating with value M. No value greater than \$FFFF will be generated. Each time that a sequence of length L is generated, N and M are incremented by I. If N and M are to remain constant, I must be zero. Otherwise the limits of the ascending binary sequences (N through M) will be altered in the manner described above. An example of the use of data generator "A" would be: DATA1 A 1 5 0 10. A DATA1 parameter in an OUTPUT statement would now specify that the sequence 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, be output to the named peripheral. A DATA1 parameter in an INPUT statement would now specify that 10 words of data be input from the named peripheral. A DATA1 parameter in a CHECK statement would now specify that the first ten words input by the most previous INPUT statement be checked against the sequence 1, 2, 3, 4, 5, 1, 2, 3, 4, 5.
DATAi B N P Q R S	Data generator "B" generates the values of parameters P Q R S . . . sequentially, N times. Parameters must have values falling in the range 0 to \$FFFF. The maximum number of parameters which may be specified is a function of the total number of parameters specified in previous DATAi statements. In total, the user may specify up to 40 DATAi parameters.

Figure 24-4. Symbolic Character Designations for Data Generators (Sheet 1)

Symbolic Character/ Parameters	Definitions
	An example of the use of data generator "B" would be: DATA2 B 3 5 4 3 2 1. A DATA2 parameter in an OUTPUT statement would now specify that the sequence 5, 4, 3, 2, 1, 5, 4, 3, 2, 1, 5, 4, 3, 2, 1 be output to the named peripheral. A DATA2 parameter in an INPUT statement would now specify that 15 words of input data be input from the named peripheral. A DATA2 parameter in a CHECK statement would now specify that the first fifteen words input by the most previous INPUT statement be checked against the sequence 5, 4, 3, 2, 1, 5, 4, 3, 2, 1, 5, 4, 3, 2, 1.

Figure 24-4. Symbolic Character Designations for Data Generators (Sheet 2)

expected data. Both the format of the data and the total number* of characters input are specified by the DATAi statement. The data is not checked as to format during the execution of this statement.

An example of this statement might be:

INPUT TTYK DATA5

This causes data as specified by another statement (DATA5) to be input from the teletype keyboard.

*The maximum number of data words which may be input by a given INPUT statement may be determined by examining the POSY listing. The maximum is equal to the number of bytes of memory above the symbolic address "ZZ", divided by two. In any event, all systems should enable the user to input at least 80 words with each INPUT statement.

OUTPUT name DATAi.

The execution of this statement causes the named peripheral to output data as specified by the DATAi field.

The DATAi field is as previously defined.

An example of this statement might be:

OUTPUT LINP DATA0.

This causes data as specified by another statement (DATA0) to be output to the line printer.

CHECK DATAi.

The execution of this statement causes data input by the preceding INPUT statement to be checked against criteria established by the specified DATAi statement.

peripheral operating system (POSY)

A word for word comparison is made between the previously input data and the expected data (as specified by the referenced DATAi statement).

At the conclusion of the test, a standard format printout of the outcome occurs (on sense-switch option). All encountered errors are reported. If no errors are encountered, nothing will be printed out.

If Sense Switch 2 is set, however, all printout is suppressed. Under this option, the user may examine location \$0035 via the console at the completion of his test. If a CHECK statement has detected one or more errors, the contents are set to \$FF. Otherwise, the contents are \$00.

Each time the user enters a new test program, the contents at location \$0035 will be set to \$00.

An example of this statement might be:

```
CHECK DATA1.
```

This causes the block of data that was input by the previous INPUT statement to be checked against the data generated by the referenced DATA1 statement.

An example of a CHECK statement error message would be:

DEVICE	ERROR	E.V.	A.V.
\$0005	DATA	\$0FFF	\$0FFE

The interpretation of the above example is:

"A data error occurred. The peripheral corresponding to device number \$0005 input the value \$0FFE (actual value —

A.V.) but the expected value (E.V. — as defined by the CHECK statement) was \$0FFF."

LOOP n m.

The execution of this statement causes the program to transfer to the statement marked by the nth preceding asterisk for m successive times.

Execution then continues with the statement following the LOOP statement.

An example of this statement might be:

```
---  
---  
*ASSIGN CRDR A 5.  
*DATA2 B 40 $AAA $555.  
*INPUT CRDR DATA2.  
*CHECK DATA2.  
*LOOP 2 100.  
---  
---
```

The LOOP statement causes the INPUT and CHECK statements to be executed 100 times in addition to the first time through. Upon completion, the program continues with whatever statement follows the LOOP statement.

CONTROL command.

The nature of this statement is determined by the command field.

If the command is within the range \$3300 to \$33FF, the command is taken to be a meaningful function command and the command is executed when the CONTROL statement is executed. The entire command

must be explicitly defined, including the operation code, the order code and the device address. The available command fields are as given in Figure 24-5.

An example of this statement might be:

CONTROL \$33C1.

This causes the teletype paper tape reader to read one character.

Alternatively, if the command is outside the range \$3300 to \$33FF, the command is taken to be a means of controlling the performance of an immediately following INPUT or OUTPUT statement. Acceptable commands of this sort are also given in Figure 24-5.

An example of this usage of a CONTROL statement might be:

CONTROL \$1000.

This causes an immediately following INPUT statement, which specifies TTYP, to input paper tape under the read-continuous mode rather than under the standard read-one-character-at-a-time mode of operation.

PAUSE n.

The execution of this statement causes the test program to halt.

Pressing RUN on the computer console continues the execution of the test program n preceding statements, or if n is equal to zero, the execution is continued at the very next following statement. The n field determines the preceding statements to which control will be returned upon continuation. Execution continues with the statement

marked by the nth preceding asterisk. An alternative path causing control to return to the compiler mode of operation may be activated by setting Sense Switch 3.

An example of this statement might be:

PAUSE 6.

This causes the test program to halt, and when continued, to start execution at the sixth previous accepted statement.

BEGIN.

The presence of this statement causes the compiling operation to cease and control to be given to the test program.

Execution of the test program begins at the first executable statement which was input to the compiler.

Preparation of POSY Test Programs

A test program consists of a sequence of POSY statements defining a specific input/output operation involving one or more peripherals.

Each peripheral to be activated must be defined by an ASSIGN statement prior to any reference to that peripheral by an INPUT or OUTPUT statement.

A peripheral may be defined by more than one ASSIGN statement in the course of a program. INPUT/OUTPUT statements referring to that peripheral will utilize the definition provided by the most recent ASSIGN statement.

EXECUTABLE COMMANDS*

Card Reader (Assuming device address 5)

\$3345 Read a Card

Teletype (Assuming device address 1)

\$3321 Reset Teletype
\$3341 Read Paper Tape — Continuous
\$33C1 Read Paper Tape — One Character

High Speed Paper Tape Reader (Assuming device address 2)

\$3322 Start Reader
\$33A2 Stop Reader

NON-EXECUTABLE COMMANDS

Teletype Paper Tape Reader

\$1000 Employ the "read continuous" function when
 executing the following INPUT statement.

High Speed Paper Tape Reader

\$1100 Employ the "read continuous" function when
 executing the following INPUT statement.

Line Printer

0 Paper feed controlled by tape channel 0 when
 executing the following OUTPUT statement.
1 Paper feed controlled by tape channel 1
2 Paper feed controlled by tape channel 2
3 Paper feed controlled by tape channel 3
4 Paper feed controlled by tape channel 4
5 Paper feed controlled by tape channel 5
6 Paper feed controlled by tape channel 6
7 Paper feed controlled by tape channel 7

Figure 24-5. POSY CONTROL Statement Command Fields (Sheet 1)

NON-EXECUTABLE COMMANDS (Cont)Line Printer

8	Paper feed stepped no lines
9	Paper feed stepped one line
10	Paper feed stepped two lines
11	Paper feed stepped three lines
12	Paper feed stepped four lines
13	Paper feed stepped five lines
14	Paper feed stepped six lines
15	Paper feed stepped seven lines

*Any valid FUNCTION, SENSE, BTI, or BTO command may be specified by a CONTROL statement. The above commands are given only as examples.

Figure 24-5. POSY CONTROL Statement Command Fields (Sheet 2)

Any reference to a DATAi statement must have been previously defined by the corresponding DATAi statement.

The last executable statement should be a PAUSE n., where n is non-zero, to cause a normal halt upon completion of execution of the test program.

Each test program must be terminated by a BEGIN statement to cause execution to start.

In constructing a POSY test program, one typically assigns the applicable device addresses and modes of all the peripherals involved in the test by inputting the necessary ASSIGN statements.

Next, the type of data desired for the test may be selected from Figure 24-4. The DATAi statements are then input, specifying

ing the type of data patterns desired and the controlling parameters.

The INPUT/OUTPUT statements are then input as desired, along with possible CHECK statements to have the input data checked, and possible LOOP statements to cause the test to be repeated a desired number of times.

Upon the completion of the execution of the test, a PAUSE statement with a non-zero parameter should be included to maintain control over the program.

The last statement to be input must be a BEGIN statement in order to start execution of the test program.

The following example illustrates a test program and what it accomplishes:

*ASSIGN CRDR D 5 1.

*DATA0 B 40 \$AAA \$555.

peripheral operating system (POSY)

```
*INPUT CRDR DATA0.  
*CHECK DATA0.  
*LOOP 2 100.  
*PAUSE 3.  
*BEGIN.
```

The ASSIGN statement identifies the card reader as being device number 5. The card reader is to be tested under interrupt mode and has interrupt line 1 associated with it. The data involved in this test is defined by the DATAi statement as being type B. This data generator defines the required data to consist of the pattern \$AAA \$555 to be repeated 40 times. This represents 80 columns of alternate punched rows on each card. The INPUT statement causes a card to be read into an input buffer. The entire card is read because the corresponding DATAi statement specifies 80 data words. The CHECK statement compares all 80 data words previously input with the specified alternate bit pattern. If any discrepancies are found they are reported to the teletype in a standardized printout format. The LOOP statement causes the INPUT, CHECK and LOOP statements to be executed an additional 500 times. Upon completion, the PAUSE statement is executed, which causes the test program to halt. Upon pressing RUN, the entire test program is repeated (or returned to the compiler mode of operation on sense switch option).

Operating the POSY System

The POSY object paper tape is loaded by using the Binary Loader (Section 24).

The user then loads the particular object tape associated with the peripheral he is interested in testing. The identification

numbers associated with the various object tapes are given in Figure 24-6.

Users need to load the POSY object tape only once. They do, however, need to load a new "peripheral" object tape whenever they wish to proceed from the testing of one peripheral to another.

After the POSY object tape is loaded, along with a "peripheral" object tape, all sense switches should be placed in the reset position and the POSY program started at \$0020.

The program responds by outputting the header POSY and halting (P=\$3A). The user then selects sense switch options as follows:

Sense Switch	Definition
1 reset	Keyboard input of test program
1 set	Teletype paper tape input of test program

Sense switch 1 may be set or reset at any time during the input of the source language test program. Sense switches 2 and 3 should be reset at this time.

Sense switch 2 is used to provide for error message suppression during the execution of the test program.

Sense switch 3 should be reset at this time. Sense switch 3 is used to select between options available when halts occur during the execution of the test program. Such options are described in Figure 24-7.

Identification Numbers	Description
92A0103-060	POSY (includes no peripherals)
92A0103-061	Card Reader Over-layer Routine
92A0103-062	High Speed Paper Tape System Over-layer Routine
92A0103-063	Line Printer Over-layer Routine
92A0103-064	Teletype Over-layer Routine

Figure 24-6. Identification Numbers of POSY Object Tapes

While the computer is halted, the user must prepare each peripheral which is to be tested (e.g., POWER ON). Then SYSTEM RESET must be pressed on the Varian 520/i console.

Test programs entered via paper tape should:

1. be known to be error-free.
2. should not contain carriage return or line feed characters (use RUBOUTs), and
3. should begin with "PAUSE 0".

Having selected the desired sense switch settings, depress RUN to cause the source statements to be input.

When the system processes a BEGIN statement, control is transferred to the generated test program and the various peripherals are exercised as requested.

Once the execution of the test program has begun, SYSTEM RESET must not be depressed at any time prior to completion of

the test program, unless specifically directed otherwise by error halt codes (Figure 24-7).

Error messages may occur during the compilation of a source program or during its execution. The error messages that may occur during compilation are listed in Figure 24-8. The presence of any of these errors indicates that the previous source statement could not be completely processed and must be re-entered correctly.

The error messages that occur during the execution of the test program generally indicate a condition of the tested peripherals that does not allow the test to continue (e.g., unit not ready). For these errors, the computer is halted following the message timeout to allow the condition to be corrected. Before halting, though, a code is loaded into the C register to identify the error condition. The C register may be examined via the console and the code may be interpreted with Figure 8-7. Descriptions of the error conditions and recovery techniques are detailed in this table.

C Register	Definition
\$0001	Prepare to enter program. Select sense switch options: SS1 Reset: Keyboard Input. SS2 Set: Paper Tape Input Press SYSTEM RESET Press RUN and enter program.
\$0002	PAUSE statement with a non-zero N parameter has been encountered. Select sense switch option: SS3 Reset: Branch Back to the Specified Statement SS3 Set: Return to Compilation Mode Press RUN.
\$0003	PAUSE statement with a N parameter of zero has been encountered. Select sense switch option: SS3 Reset: Continue with Next Statement SS3 Set: Return to Compilation Mode Press RUN.
\$0004	Illegal MODE. INPUT/OUTPUT was attempted with a peripheral not yet implemented for operation in the assigned MODE. Program must be corrected and re-entered. Press RUN to return to compilation mode.
\$0009	Teletype paper tape reader timed out. An attempt to SENSE INPUT BUFFER READY did not receive a true response within 300 milliseconds. Select sense switch option: SS3 Reset: Try again to Read the Record. (Position the tape at the beginning of the incompletely read record; i.e., at the beginning of that portion of the tape which was in the process of being read when the error occurred.) SS3 Set: Return to Compilation Mode Press RUN.

Figure 24-7. POSY Halt Codes and Their Descriptions (Sheet 1)

C Register	Definition
\$000A	Teletype paper tape punch or page printer timed out. An attempt to SENSE OUTPUT BUFFER READY did not receive a true response within 300 milliseconds. Select sense switch option: SS3 Reset: Try Again to Complete Output SS3 Set: Return to Compilation Mode Press RUN.
\$0010	Illegal Exit. Last executable statement was not a PAUSE statement with a non-zero parameter. Select sense switch option: SS3 Reset: Repeat Test Program SS3 Set: Return to Compilation Mode Press RUN.
\$0017	Interrupt error. Same error as was previously reported (\$0026). Before recovery can proceed, SYSTEM RESET must be pressed on the 520/i console. Now press RUN and control will return to the <u>start</u> of your test program.
\$0020	Card reader timed out. SENSE HOPPER EMPTY was false, SENSE READER ERROR was false, but an attempt to SENSE UNIT READY did not receive a true response within 100 milliseconds. Display Q register. If \$FFFF, the last card through the reader must be re-read. Place the last card through the reader at the bottom of the input hopper so that it will be the next card to be read. If the Q register is \$0000, <u>do not</u> move the last card through the reader; simply proceed. Select sense switch option: SS3 Reset: Try again to read the card. SS3 Set: Return to compilation mode. Press RUN.

Figure 24-7. POSY Halt Codes and Their Descriptions (Sheet 2)

C Register	Definition
\$0021	Card reader hopper empty. SENSE HOPPER EMPTY was true. Select sense switch option: SS3 Reset: Try again. (Place cards in hopper.) SS3 Set: Return to compilation mode. Press RUN.
\$0022	Card reader error. SENSE HOPPER EMPTY received a false response and SENSE READER ERROR received a true response. Display Q register. If \$FFFF, the last card through the reader must be re-read. Place the last card through the reader at the bottom of the input hopper so that it will be the next card to be read. If the Q register is \$0000, do not move the last card through the reader; simply proceed. Select sense switch option: SS3 Reset: Try again to read the card. SS3 Set: Return to compilation mode. Press RUN.
\$0023	Card reader timed out. SENSE UNIT READY came true, READ ONE CARD was commanded, but an attempt to SENSE CHARACTER READY did not receive a true response within 100 milliseconds. Display Q register. If \$FFFF, the last card through the reader must be re-read. Place the last card through the reader at the bottom of the input hopper so that it will be the next card to be read. If the Q register is \$0000, <u>do not</u> move the last card through the reader; simply proceed. Select sense switch option: SS3 Reset: Try again to read the card. SS3 Set: Return to compilation mode. Press RUN.

Figure 24-7. POSY Halt Codes and Their Descriptions (Sheet 3)

C Register	Definition
\$0024	Card reader timed out. SENSE UNIT READY came true, READ ONE CARD was commanded, but an attempt to SENSE CHARACTER READY did not receive a true response within 100 milliseconds. Display Q register. If \$FFFF, the last through the reader must be re-read. Place the last card through the reader at the bottom of the input hopper so that it will be the next card to be read. If the Q register is \$0000, <u>do not</u> move the last card through the reader; simply proceed. Select sense switch option: SS3 Reset: Try again to read the card. SS3 Set: Return to compilation mode. Press RUN.
\$0026	Interrupt error. The peripheral corresponding to the output device number either interrupted on an invalid interrupt line or interrupted when its interrupt capability should have been disabled, or was interrupted by some other peripheral whose interrupts should have been disabled, or was the last device to be functioning prior to some other invalid interrupt. Display A register for the results of a Copy Interrupt Status (CIS) instruction (if applicable). Select sense switch option: SS3 Reset: Try to recover and proceed. SS3 Set: Return to compilation mode. Press RUN.
\$0030	Line printer timed out. An attempt to SENSE BUFFER READY did not receive a true response within 800 milliseconds. Select sense switch option: SS3 Reset: Try again to output the line. SS3 Set: Return to compilation mode. Press RUN.

Figure 24-7. POSY Halt Codes and Their Descriptions (Sheet 4)

C Register	Definition
\$0040	High speed paper tape reader timed out. An attempt to SENSE TRANSFER READY did not receive a true response within 100 milliseconds. Select sense switch option: SS3 Reset: Try again to read the record. (Position the tape at the beginning of the incompletely read record; i.e., at the beginning of that portion of the tape which was in the process of being read when the error occurred.) SS3 Set: Return to compilation mode. Press RUN.
\$0041	High speed paper tape punch timed out. An attempt to SENSE TRANSFER READY did not receive a true response within 100 milliseconds. Select sense switch option: SS3 Reset: Try again to complete output. SS3 Set: Return to compilation mode. Press RUN.

Figure 24-7. POSY Halt Codes and Their Descriptions (Sheet 5)

Message	Definition
COMMAND	The <u>command</u> parameter in this Control statement did not correspond to any of the acceptable <u>commands</u> as defined in Figure 24-5.
DATA	The <u>DATAi</u> parameter in this input, Output, or Check statement was illegal or the "i" in this <u>DATAi</u> statement was unacceptable. The "i" may have exceeded the acceptable range to 0 to 9, or the <u>DATAi</u> parameter may not have been previously defined by a <u>DATAi</u> statement.
DEVICE	The <u>device</u> parameter in this Assign statement did not fall within the acceptable range to 0 to \$1F.
INTERRUPT LINE	The <u>interrupt line</u> parameter in this Assign statement was not acceptable.
M FIELD	The <u>M</u> parameter in this Loop statement did not fall within the acceptable range of 0 to \$7FFF.
MODE	The <u>mode</u> parameter in this Assign statement did not correspond to any of the acceptable <u>modes</u> as defined in Figure 24-3.
N FIELD	The <u>N</u> parameter in this Pause or Loop statement did not fall within the acceptable range of 0 to X, where X represents the total number of starred statements in the user's program (excluding this Pause or Loop statement).
NO ROOM	Generation of object code for this statement would have resulted in destruction of POSY code. Either the user has attempted to include too many statements in his program, or he has attempted to use too many parameters in his <u>DATAi</u> statements. Program must be re-entered.
OPERATION CODE	The operation <u>code</u> in this statement did not correspond to any of the acceptable <u>codes</u> as defined in Figure 24-1.
PARAMETER	The actual <u>parameters</u> in this <u>DATAi</u> statement did not constitute a valid set. Fewer than two <u>parameters</u> may have been provided, or a <u>parameter</u> may not have fallen within the acceptable range of \$0000 to \$FFFF.
PERIPHERAL NAME	The <u>peripheral name</u> in this Input, Output, or Assign statement did not correspond to any of the acceptable <u>names</u> as defined in Figure 24-2.
TYPE	The <u>type</u> parameter in this <u>DATAi</u> statement did not correspond to any of the acceptable <u>types</u> as defined in Figure 24-4.

Figure 24-8. Summary of POSY Error Messages Occurring During Compilation

peripheral operating system (POSY)

Another type of message that may occur during the execution of the test program results from CHECK statements. If a CHECK statement discovers no errors in checking the data input by the previous INPUT statement, no messages are output. If errors are discovered, a message will be typed for each error. An example of such a message would be:

DEVICE	ERROR	E.V.	A.V.
0002	DATA	0001	0000

The upper line is a header which describes the format of the error message or messages which follow it below.

The interpretation of the above example is:

"A data error occurred. The peripheral corresponding to device number \$0002 input the value \$0000 (actual value — A.V.) but the expected value (E.V. — as defined by the CHECK statement) was \$0001."

If error message suppression was selected via sense switch two during execution of the test program, the user may examine location \$0035 via the console at the completion of his test. If any CHECK statement detected one or more errors, the contents will be set to \$FF. Otherwise, the contents will be \$00. Each time a new test program is entered, the contents will be reset to \$00.

I/O Interface With Peripherals

The effects of the INPUT/OUTPUT operation codes vary somewhat depending on the peripheral. The following paragraphs describe these for each specific peripheral. In

addition, the applicable CONTROL commands are given for each peripheral.

Teletype Paper Tape Reader/Punch (TTYTP). Executing an INPUT statement from the teletype paper tape reader causes 8-bit characters (the number of which is determined by the DATAi parameter) to be input. Executing an OUTPUT statement to the paper tape punch causes 8-bit characters (as specified by the DATAi parameter) to be output.

Characters are input via the read-one-character function, unless the INPUT statement is preceded by the following CONTROL statement, which specifies use of the read-continuous function:

CONTROL \$1000.

Teletype Keyboard/Page Printer (TTYK). Executing an INPUT statement from the teletype keyboard causes 8-bit characters (the number of which is determined by the DATAi parameter) to be input. Characters input from the teletype keyboard are immediately echo-printed on the teletype page printer. Execution of an INPUT statement from the teletype keyboard terminates only when the number of characters specified by the DATAi parameters has been input.

Executing an OUTPUT statement to the teletype page printer causes 8-bit characters (as specified by the DATAi parameter) to be output.

There are no applicable CONTROL commands associated with the teletype keyboard/page printer.

High Speed Paper Tape Reader (HSPT) Executing an INPUT statement from the high speed paper tape reader causes 8-bit characters to be input. The DATAi parameter determines the number of characters to be input.

Characters are input via the "read continuous" function if mode A is assigned. Characters are input via the "read one character" function if mode B is assigned, unless the INPUT statement is preceded by the following CONTROL statement, which specifies use of the "read continuous" function:

CONTROL \$1100.

High Speed Paper Tape Punch (HSPT) Executing an OUTPUT statement to the high speed paper tape punch causes 8-bit characters to be output. The DATAi parameter determines the number of characters to be output. There are no applicable CONTROL commands associated with the high speed paper tape punch.

Line Printer (LINP). Executing an OUTPUT statement to the line printer causes a line of 8-bit characters (as specified by the DATAi parameter — 132 characters maximum) to be output.

Paper feed control is under the step-one-line mode, unless the OUTPUT statement is preceded by a CONTROL statement with one of the following command parameters:

- 0 Paper feed controlled by tape channel 0.
- 1 Paper feed controlled by tape channel 1.

- 2 Paper feed controlled by tape channel 2.
- 3 Paper feed controlled by tape channel 3.
- 4 Paper feed controlled by tape channel 4.
- 5 Paper feed controlled by tape channel 5.
- 6 Paper feed controlled by tape channel 6.
- 7 Paper feed controlled by tape channel 7.
- 8 Step no lines.
- 9 Step one line.
- 10 Step two lines.
- 11 Step three lines.
- 12 Step four lines.
- 13 Step five lines.
- 14 Step six lines.
- 15 Step seven lines.

Paper Tape Punch/Page Printer (Teletype)

```

ASSIGN      TTYPE  A    1    X.
DATA1      A 0 $FF 0 256.
OUTPUT      TTYPE  DATA1.
PAUSE      1.
BEGIN.
```

This program assumes that the teletype is device number 1.

If the teletype is functioning properly, this program causes the paper tape punch to punch, under sense mode, a sequence of ascending binary words

peripheral operating system (POSY)

starting with 0 and terminating with \$FF.

At the same time, due to the nature of the ASR 33 TC teletype, the program also causes the page printer to print the following:

```
!''#$$%&'( )*+,-./0123456789;<=>  
?@ABCDEFGHIJKLMN O PQRSTU V  
W XYZ[ \ ] ↑←@ABCDEFD
```

```
!''#$$%&'( )*+,-./0123456789;<=>  
?@ABCDEFGHIJKLMN O PQRSTU V  
W XYZ[ \ ] ↑←@ABCDEFD
```

If the teletype is not functioning properly, error messages describing the problem are output to the teletype and the computer halts with a code in the C Register which further references the problem (see Figure 24-7).

When the tape is successfully punched, the PAUSE statement would be executed, and a halt occurs with \$0002 in the C register.

Pressing RUN with Sense Switch 3 reset causes the program to be repeated.

Pressing RUN with Sense Switch 3 set returns control to the compilation mode so that a new test program can be entered.

Paper Tape Reader (Teletype)

```
ASSIGN TTYP A 1 X.  
DATA1 A 0 $3F 0 64.  
DATA2 A $40 $7F 0 64.  
DATA3 A $80 $BF 0 64.
```

```
DATA4 A $C0 $FF 0 64.  
INPUT TTYP DATA1.  
CHECK DATA1.  
INPUT TTYP DATA2.  
CHECK DATA2.  
INPUT TTYP DATA3.  
CHECK DATA3.  
INPUT TTYP DATA4.  
CHECK DATA4.  
PAUSE 8.  
BEGIN.
```

This program assumes that the teletype is device number 1.

A tape punched with ascending binary words starting with 0 and terminating with \$FF must previously have been prepared. The tape must be positioned in the reader so that the word with the value 0 is the first to be read.

If the teletype reader is functioning properly, the program causes the tape to be read, under sense mode, and the data input to be checked against the expected binary sequence (0 through \$FF).

If data discrepancies are found by the CHECK statements, data error messages will be output.

If the teletype reader is not functioning properly, error messages describing the problem are output to the teletype and the computer halts with a code in the C Register which further references the problem (see Figure 24-7).

The PAUSE statement would then be executed and a halt would occur with \$0002 in the C Register. Pressing RUN with Sense Switch 3 reset causes the program to be repeated. Pressing RUN with Sense Switch 3 set returns control to the compilation mode so that a new test program can be entered. (The input of the tape is broken into four parts so that the capacity of POSY's input buffer is not exceeded.)

Keyboard (Teletype)

```
ASSIGN TTYK A 1 X.
DATA8 B 1 $8D $8A.
DATA9 A $A1 $DA 0 58.
INPUT TTYK DATA8.
CHECK DATA8.
PAUSE 0.
INPUT TTYK DATA9.
CHECK DATA9.
PAUSE 5.
BEGIN.
```

This program assumes that the teletype is device number 1.

The user begins by inputting a carriage return (\$8D) and a line feed (\$8A) from the keyboard.

If the teletype is not functioning properly, error messages describing the problem are output to the teletype and the computer halts with a code in the C Register which further references the problem (see Figure 24-7).

The data input would be checked and, if discrepancies were found, data error messages would be output.

A halt would then occur with \$0003 in the C Register.

The user would then press RUN and input the sequence of ASCII characters corresponding to the sequence \$A1 to \$DA:

```
!'#$%&'()*+,-./0123456789:;<=>
?@ABCDEFGHIJKLMNPOQRSTUVWXYZ
WXYZ
```

This data would be checked and, if discrepancies were found, data error messages would be output.

Then the second PAUSE statement would be executed and a halt would occur with \$0002 in the C Register.

To repeat the program, RUN would be pressed with Sense Switch 3 reset.

To return to the compilation mode, RUN would be pressed with Sense Switch 3 set.

High-Speed Paper Tape Punch

```
ASSIGN HSPT A 2 X.
DATA7 A 0 $FF 0 256.
OUTPUT HSPT DATA7.
PAUSE 1.
BEGIN.
```

This program assumes that the high speed paper tape system is connected on the 8-bit I/O bus as device number 2.

peripheral operating system (POSY)

If the paper tape punch is functioning properly, the program causes a tape consisting of a sequence of ascending binary words starting with 0 and terminating with \$FF to be punched, under sense mode.

If the paper tape punch is not functioning properly, error messages describing the problem are output to the teletype and the computer halts with a code in the C Register which further references the problem (see Figure 24-7).

At the completion of the successful output of the sequence, the PAUSE statement is executed and a halt occurs with \$0002 in the C Register.

To repeat the program and punch another sequence, the user presses RUN with Sense Switch 3 reset.

To return to the compilation mode so that a new test program can be entered, RUN would be pressed with Sense Switch 3 set.

High-Speed Paper Tape Reader

```
ASSIGN HSPT A 2 X.  
DATA3 A 0 $3F 0 64.  
DATA4 A $40 $7F 0 64.  
DATA5 A $80 $BF 0 64.  
DATA6 A $C0 $FF 0 64.  
INPUT HSPT DATA3.  
CHECK DATA3.  
INPUT HSPT DATA4.  
CHECK DATA4.  
INPUT HSPT DATA5.  
CHECK DATA5.
```

```
INPUT HSPT DATA6.  
CHECK DATA6:  
PAUSE 8.  
BEGIN.
```

This program assumes that the high speed paper tape system is connected on the 8-bit I/O bus as device number 2.

A tape punched with ascending binary words starting with 0 and terminating with \$FF must previously have been prepared. The tape must be positioned in the reader so that the word with the value 0 is the first to be read.

If the high speed paper tape reader is functioning properly, the program causes the tape to be read, under sense mode, and the data input to be checked against the expected binary sequence (0 through \$FF).

If data discrepancies are found by the check statements, data error messages will be output.

If the high speed paper tape reader is not functioning properly, error messages describing the problem are output to the teletype and the computer halts with a code in the C Register which further references the problem (see Figure 24-7).

The PAUSE statement would then be executed and a halt would occur with \$0002 in the C Register.

To repeat the program, the user presses RUN with Sense Switch 3 reset.

To return to the compilation mode so that a new test program can be entered, RUN is

pressed with Sense Switch 3 set. (The input of the tape is broken into four parts so that the capacity of POSY's input buffer is not exceeded.)

Card Reader

```
ASSIGN CRDR A 5 X.
DATA1 B 40 $AAA $555.
INPUT CRDR DATA1.
CHECK DATA1.
LOOP 2 99.
PAUSE 3.
BEGIN.
```

This program assumes that the card reader is connected on the 8-bit I/O bus as device number 5.

If the card reader is functioning properly, the program causes 100 cards to be read, under sense mode.

The data input from each card is checked against the specified data pattern (rows 12, 0, 2, 4, 6, and 8 punched in the odd columns; rows 11, 1, 3, 5, 7, and 9 punched in the even columns). Any discrepancies in the data are output to the teletype.

If output of the outcome of each execution of the CHECK statements is not desired, Sense Switch 2 may be set and all messages are suppressed.

If the card reader is not functioning properly, error messages describing the problem are output to the teletype and the computer halts with a code in the C Register which further references the problem (see Figure 24-7).

At the completion of successful input of 100 cards, the PAUSE statement is executed and a halt occurs with \$0002 in the C Register.

To repeat the program and read another 100 cards, the user presses RUN with Sense Switch 3 reset.

To return to the compilation mode so that a new test program can be entered, RUN would be pressed with Sense Switch 3 set.

Line Printer

```
ASSIGN LINP A 7 X.
DATA2 A $A0 $DF 0 132.
CONTROL 1.
OUTPUT LINP DATA2.
LOOP 1 49.
PAUSE 3.
BEGIN.
```

This program assumes that the line printer is connected on the 8-bit I/O bus as device number 7.

If the line printer is functioning properly, the program causes it to go to "top of page" (Tape Channel 1) and print 50 single-spaced lines under sense mode. Each line contains 132 characters, consisting of the following concatenated ASCII sequences:

(\$A0 through \$DF) +
(\$A0 through \$DF) +
(\$A0 through \$A3).

If the line printer is not functioning properly, error messages describing the problem are output to the teletype and the computer halts with a code in the C Register

peripheral operating system (POSY)

which further references the problem (see Figure 24-7).

At the completion of the successful output of the 50 lines, the PAUSE statement is executed and a halt occurs with \$0002 in the C Register.

To repeat the program and output another 50 lines, the user would press RUN with Sense Switch 3 reset.

To return to the compilation mode so that a new test program can be entered, RUN would be pressed with Sense Switch 3 set.

SECTION 25 – INSTRUCTION ADDRESS BOUNDARY TEST

The Instruction Address Boundary Test is designed to test all classes of machine instructions at a critical memory address boundary to see if the instruction crosses the boundary correctly.

A critical memory address is one whose least significant 8 bits are $(11111111)_2$, or \$FF in hexadecimal notation. For example, an instruction at \$FF crosses the boundary correctly if the P register is incremented at either \$100 or \$101 (depending on the length of the instruction). If the boundary is crossed successfully, the tests continue; otherwise, an error message is output.

The program, a valuable supplement to the Instruction Tests included in the Basic Computer Test Program, requires only the minimum Varian 520/i configuration, 4K memory and teletype, and does not require any additional software except the Binary Loader. Since the program and test data are scattered throughout the lower 4K bytes of the machine (see Figure 25-1), other programs in core at the time of loading will be overlaid in part, except for the Binary Loader at \$FBO.

The program executes test instructions at each critical boundary and determines if the test was successful. The test instructions and test address for each boundary are supplied to the program via test blocks. These blocks are decoded and the instructions tested as specified. The program already contains

several test blocks, but more tests may be performed by adding new test blocks.

The contents of the test blocks are given in Figure 25-2. The blocks may be added at assembly time, or manually at execution time. If they are added at assembly time, the source tape has to be updated. In addition to adding the new block, the new end-of-block list address has to be updated. This is done by increasing its value to the current end-of-the-block list.

Operating the Instruction Address Boundary Test

The Instruction Address Boundary Test is operated by the following sequence of actions:

1. Use the Binary Loader (Section 15) to load the program. If the load-and-go option is used, the test will begin. If the load-and-halt option is used the program can be started by pressing RUN or by resetting P to \$621 and pressing RUN.
2. When the program is started or restarted, it issues a carriage return and line feed to the teletype and begins the tests.
3. The program continues testing until the tests are finished or until an error is detected.

instruction address boundary test

000	DA4	0	Correct Data	0
0FF	DA4	-1	Incorrect Data	
1FE	DA2	0,0,0	Correct Data	0
2FE	DA2	0,\$1616, ER+\$E000	Error	
316	DA2	0, ER + \$E000	Error	
3FC	DA4	0,0	Current Test Block	
				400
404	DA2	OK + \$E000	Error	
4FF	DA4	\$ 00FFFFFF	Incorrect Data	
5FE	DA2	0,0,0	Correct Data	1
620	PROGRAM	AREA		
				800
900	DA4	-1	Incorrect Data	
9FF	DA4	0	Correct Data	2
A20	DATA	AREA		
D00	DA3	-1	Incorrect Data	
DFF	DA4	0	Correct Data	3
				1000

In the Memory Map given above the areas labeled "Correct Data" (zeros) are data which will be examined if no boundary failure occurs, while those areas labeled "Incorrect Data" (ones) are data which will be examined when a boundary failure does occur. The areas marked "Error" are branches to the error handler and are executed when boundary errors occur.

Figure 25-1. Memory Map, Instruction Address Boundry Test

4. When an error is detected, the program types out an error message of the following form:

```
XXXX  XX  XX  XX
  A    O   B   E
```

where:

- A is the address of the opcode or operand of the test instruction
- O is the opcode or operand of the test instruction

Byte 0:	Starting opcode or operand minus the increment
1:	Ending opcode or operand plus the increment
2:	Increment to get to the next opcode or operand
3 & 4:	Address where opcode or operand is to be stored
5 - 8:	4 bytes to be stored at \$3FC, contains test instruction
9 - 12:	4 bytes to be stored at \$400, contains code which tests for successful execution of program

Bytes 3 & 4 of	\$3FC
Test Block contain	\$3FD
one of these	\$3FE
addresses	\$3FF
	\$400
	\$401
	\$402
	\$403

(Location \$404 contains a branch to the successful completion routine)

Figure 25-2. Contents of Typical Test Block

B is the block number indicating the instruction class:			BLOCK #	INSTRUCTION CLASS	TEST LOCATION
BLOCK #	INSTRUCTION CLASS	TEST LOCATION	\$B8	LOD - AND	\$3FF
\$B0	CST - AAD	\$3FF	\$B9	SIE - SS3	\$3FE
\$B1	TAC - AAC	\$3FE	\$BA	SIE - SS3	\$3FF
\$B2	BTO - FUN	\$3FE	E is the environment:		
\$B3	BTO - FUN	\$3FF	00 Interruptable		
\$B4	BRM + CSQ	\$3FD	01 Non-interruptable		
\$B5	BRM + CSQ	\$3FE	After typing the error message the program continues if Sense Switch 1 is not set and will halt if Sense Switch 1 is set.		
\$B6	BRM + CSQ	\$3FF			
\$B7	LOD - AND	\$3FE			

instruction address boundary test

If the halt mode is selected the program will set C to zero and then halt. When RUN is pressed, the program will continue with the tests if C is still zero; otherwise it will start over from the beginning.

If the program halts or loops for several seconds without typing an error message, it is possible an instruction has failed in an unknown manner. If this should happen, the current status can be displayed by manually branching to the error routine (the error routine is labeled ER in the listing).

5. After testing is complete the program halts at the restart address (\$621).

Procedure for Determining Type of Error

The procedure for determining the type of error that has been detected is as follows:

1. Load and run the program in the halt mode (Sense Switch 1 set).
2. When the program finds an error, it types out a message and halts.

3. The error is in one of three categories:
 - a. The P register was incremented improperly which resulted in a branch.
 - b. Incorrect data was entered into the accumulator (A, C = 0 is correct).
 - c. Both of the above.
4. To determine which error occurred, the test in question can be stepped through, starting at location NX/-14. When the P register has the same address as the error address given in the error message, the instruction in question is about to be executed. (Note: In blocks B9 and BA the error address will be one greater than the actual instruction address.) When STEP is pressed again the three types of errors given above can be tested. If no registers are changed, RUN can be depressed and the program will return to where it was when the stepping process was begun. If the registers have been altered, the program can be restarted at \$621.
5. If errors are detected in BRM or STO, test data may have been altered and the program may have to be reloaded.

SECTION 26 – INSTALLATION AND INTEGRATION

Before unpacking a Varian 520/i computer system, examine cartons and crates for shipping damage.

After the equipment is unpacked, the user or installer should verify that all items have been received. Then the various system components can be installed and interconnected using the procedures outlined in this section.

System Layout and Planning

The principal components of a Varian 520/i system (central processor, memories, controllers and power supplies) are normally installed in a vertical 19-inch rack or cabinet with front and back access (VDM or equivalent user supplied). The system also normally includes an ASR-33 Teletype, mounted on its own stand or one supplied by the user.

The central processor, first 8K of memory, teletype I/O circuits, mainframe power supply and the logic section of the following options are packaged in the mainframe assembly:

Memory parity

Peripheral adapter

Punched-tape controller

Direct-memory-access port

Power fail/restart

Additional memory modules and I/O controllers are contained in one or more expansion chassis.

AN I/O controller expansion chassis may be installed above or below the mainframe. All expansion chassis containing memory modules, however, must be located above the mainframe and directly adjacent to it in order to accommodate the computer-to-memory signal cabling.

Power supplies are normally included in each expansion chassis for the memory or I/O controllers contained in the chassis. Certain combinations of controllers may require additional power, and in such cases, an auxiliary power supply is provided.

A typical installation consisting of a mainframe, two memory expansion chassis and three I/O controller expansion chassis is shown in Figure 26-1. The capacity of the system, as illustrated, is 32K of memory, up to 32 8-bit and 16-bit I/O controllers.

Cable connections are located both on the rear panel and behind the front panel. These interconnections are summarized in Figures 26-2 and 26-3.

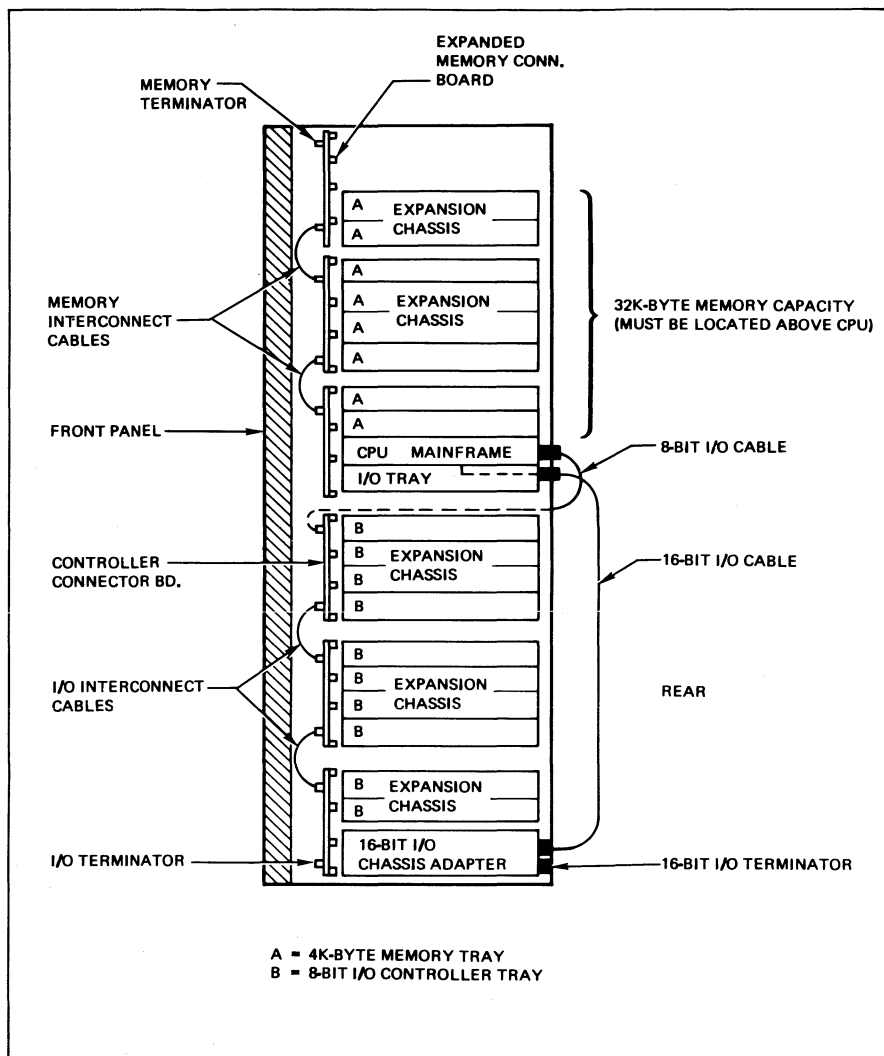


Figure 26-1. Side View of Typical System Installation

Details on all of these component elements and interconnecting cables are provided below in the following subsections:

- Teletype
- Mainframe
- Expansion Chassis
- Memory Expansion and Cabling
- 8-Bit I/O Controllers and Cabling
- 16-Bit I/O Controllers and Cabling
- Interrupt Connections
- Power Supplies

System Installation

The installation of a Varian 520/i system can be facilitated by proceeding as follows:

1. Read this complete section of the Handbook and study the system layout drawing.
2. Plan system layout and mechanically mount the Varian 520/i mainframe and any expansion chassis into cabinet or rack.
3. Install memory signal cabling (if more than an 8K system).
4. Install any slide-mounted units (VersaLOGIC, etc.).
5. Install the Varian 520/i cabling.

At this point, the user may wish to operate the central processor without peripherals. If so, perform Step 7 and return to do steps 6 and 8, if applicable.
6. Install peripheral and controller cabling.
7. Set up for Varian verification testing.
8. Proceed with entire system verification.

Teletype

The teletype unit is shipped separately from the balance of the computer and should be installed in accordance with the following manufacturer's recommendations:

1. Remove the teletype from its pallet by removing seven screws underneath the pallet.
2. Remove the LINE/OFF/LOCAL switch knob and the metal cover plate.
3. Remove four front panel screws and three thumb screws in the rear of the teletype. Remove the paper platen knob.
4. Remove the small screw on the left side of the teletype below the reader switch.
5. Carefully remove the cover.
6. Remove the wire securing the carriage assembly to the frame.
7. Remove the shipping clip from the reader assembly.

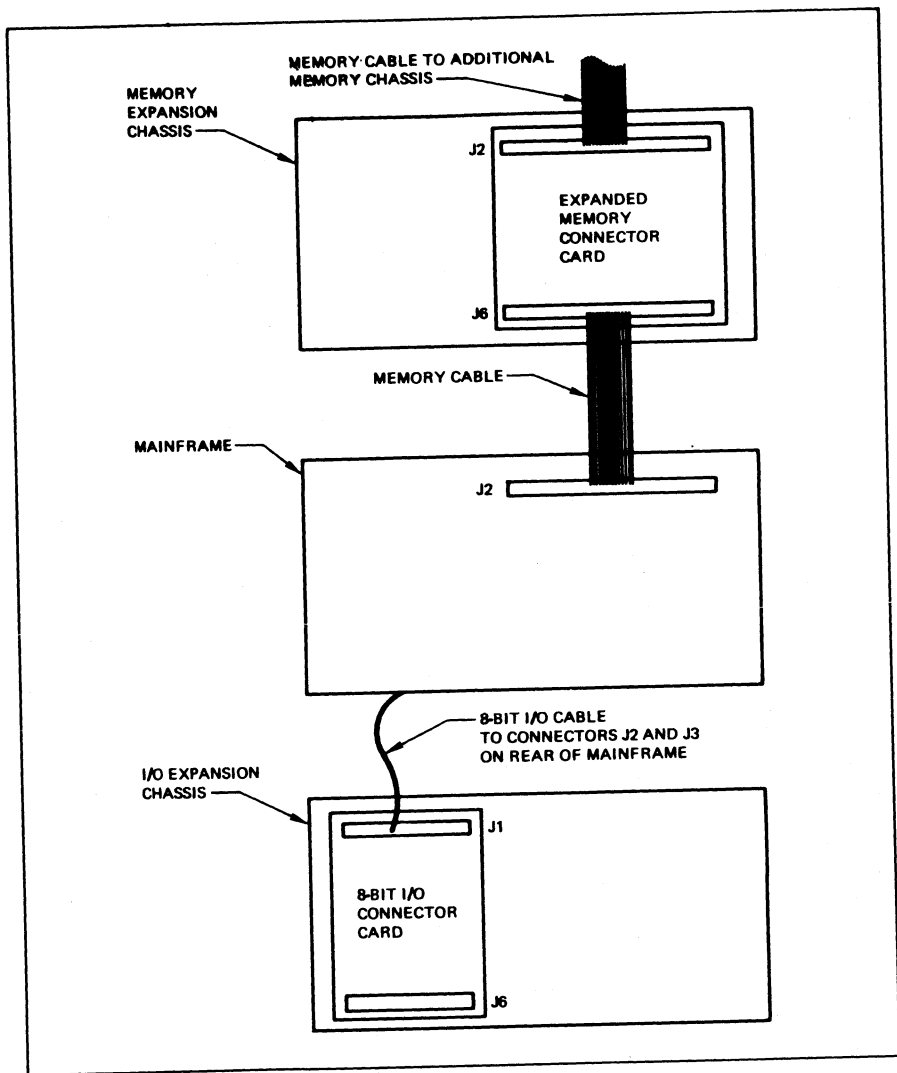


Figure 26-2. Cable Connections, Front of System (Front Panels Open)

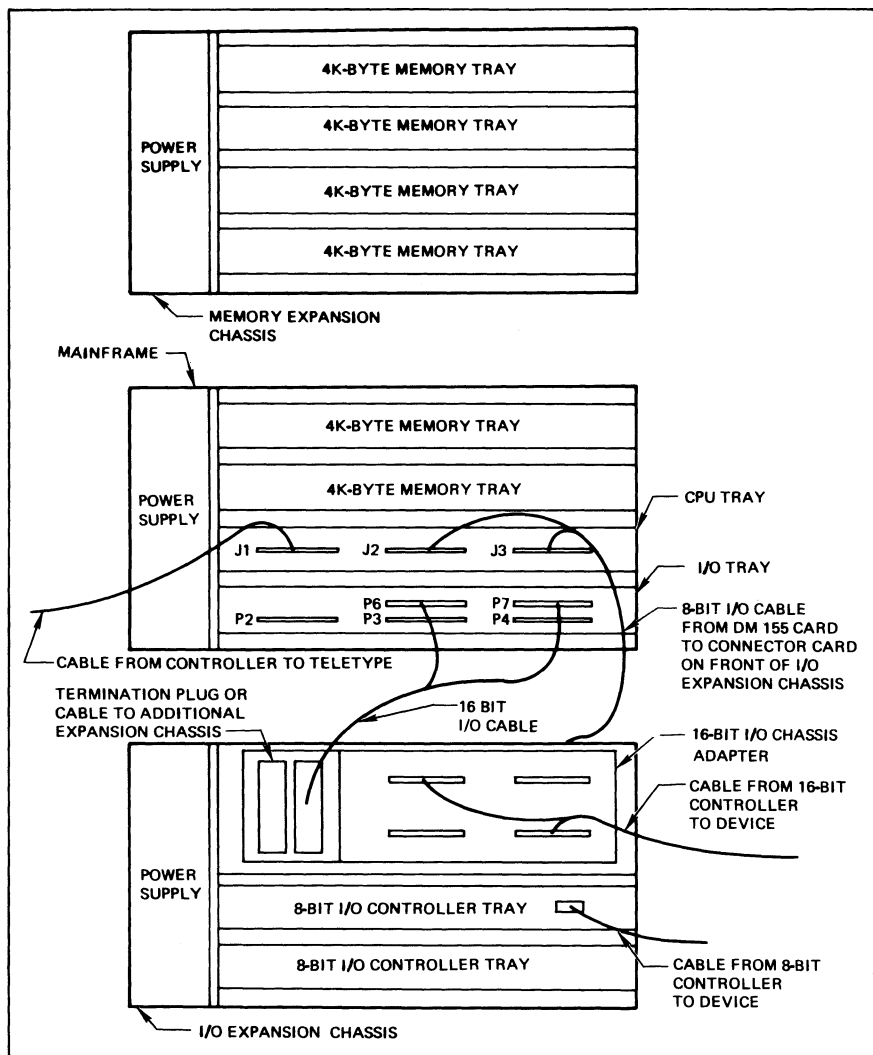


Figure 26-3. Cable Connections, Rear of System

installation and integration

8. Connect the 48-volt power supply to the short cable at the rear of the teletype. This supply is shipped in the accessory box with the chad box, etc.
9. Check to be sure that the (53C0089) cable to the computer is connected to plug 2.
10. Carefully replace the cover, being extra cautious in the area of the teletype reader.
11. Replace the front panel and all screws removed in Steps 3 and 4.
12. Replace the LINE/OFF/LOCAL switch knob removed in Step 2.
13. Connect the paper platen knob to the platen assembly.
14. Connect the computer end of the teletype/computer cable to Connector J1 on the I/O Interface Card (DM 155) in the Varian 520/i mainframe (see Figure 26-16).
15. Connect the power cable to a source of 110V, 60Hz line voltage.

Note: Retain all packing material. Also refer to the teletype repackaging instructions included in the shipping container if the teletype is reshipped.

Mainframe

The Varian 520/i mainframe contains the central processor, 4K or 8K of memory, teletype I/O circuits, and various logic ele-

ments. A minimum system configuration would consist of the main frame with 4K of memory, plus a teletype for data input/output.

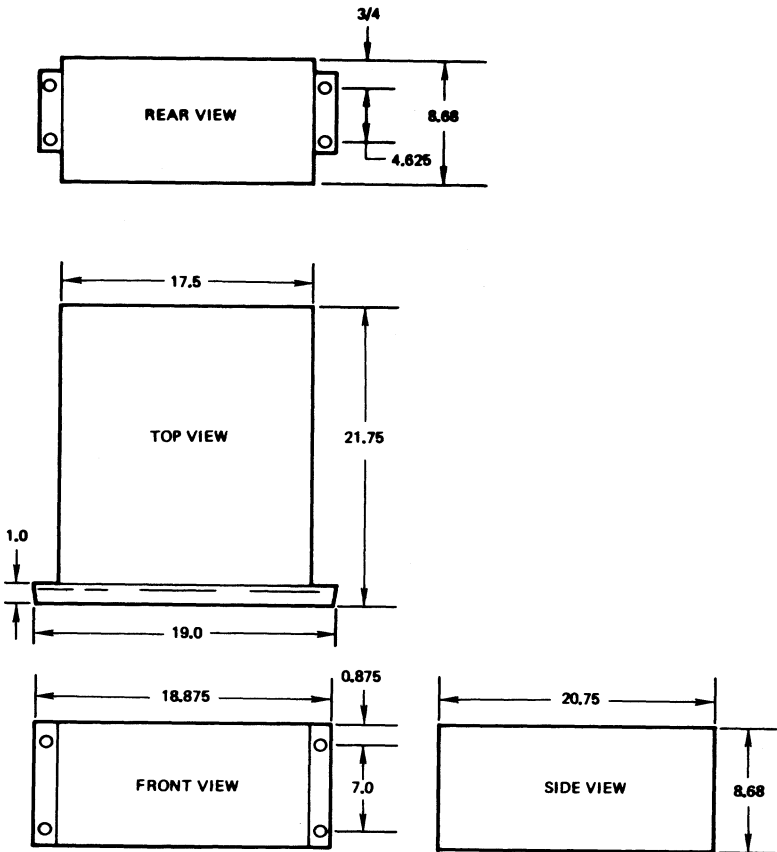
The mainframe chassis is designed for rack mounting, using standard RETMA techniques. Overall dimensions are given in Figure 26-4.

The front panel is hinged for ready access to the cable connections at the front of the chassis. The controls and indicators on the front panel are fully operable in both the open and closed positions. A description of these controls and indicators is given in Section 5.

The mainframe is equipped with two removable feet under the front end of the chassis. The feet are used to prevent the weight of the computer from being supported by the front panel whenever the unit is placed on a flat surface, such as a bench or table top. Before the mainframe is rack mounted, the feet should be removed by loosening a screw located through the center of each foot. It is recommended that the feet be stored so they may be used again if the computer is ever removed from the rack and placed, even temporarily, on a flat surface.

The internal organization of the mainframe is shown in Figure 26-5. The power supply for all mainframe elements is located on the left side, facing from the rear. The remaining space is divided into four horizontal card slots, each of which can accommodate an assembly module or "tray."

The trays are easily removed (see upper photo, back cover) by first removing a



NOTE: DIMENSIONS ARE GIVEN IN INCHES

Figure 26-4. Outline Drawings of Mainframe and Expansion Chassis

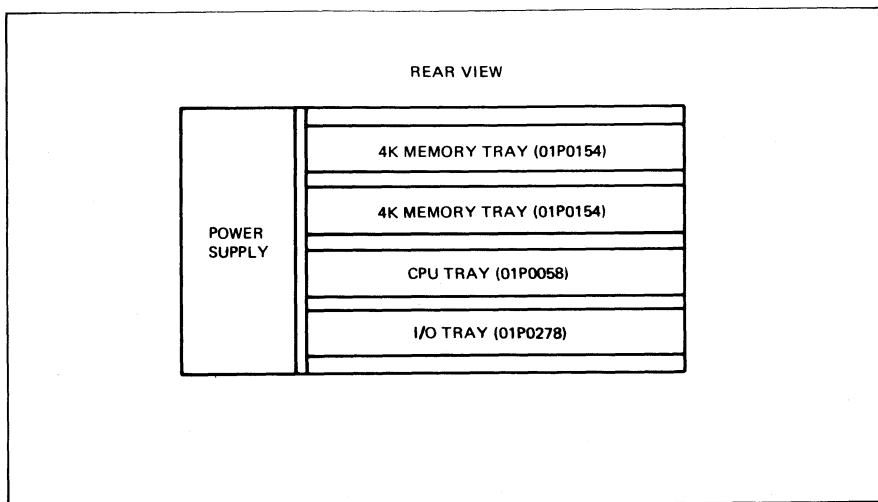


Figure 26-5. Tray Locations, Mainframe

retainer bracket on the right side of the chassis, then pulling out the tray.

Three types of trays are represented in the mainframe: a CPU (Central Processor Unit) tray, a memory tray, and an I/O tray.

A minimum Varian 520/i system would include a CPU tray and one 4K-byte memory tray. This system could be expanded by adding a second 4K-byte memory tray and/or an I/O tray carrying optional hardware elements such as a DMA card, 16-Bit Peripheral Adapter, or Paper Tape Controller.

Circuit components are mounted on individual circuit cards that plug into the three types of tray assemblies as follows:

CPU Tray P/N 01P0058

1. DM 141, Socket Card, P/N 44P0177
2. DM 154, 8 MHz Clock Card, P/N 44P0192
3. DM 155, I/O Interface Card, P/N 44P0193
4. DM 159, Memory Parity Card, P/N 44P0196 (optional)

Memory Tray, P/N 01P0154

1. DM 143, Inhibit Card, P/N 44P0189
2. DM 144, T/C and Data Card, P/N 44P0191

3. DM 145, Address Card, P/N 44P0181
4. DM 149, Core Plane Card, P/N 44P0187

I/O Tray, P/N 01P0278 (optional)

1. DM 146, Distribution Card, P/N 44P0183 (optional)
2. DM 157, Direct Memory Access Card, P/N 44P0195 (optional)
3. DM 158, Paper Tape Controller, P/N 44P0046 (optional)
4. DM 160, Peripheral Adapter Card, P/N 44P0047 (optional)

In addition to the cards listed above a DM 163 Auto Memory Disable Card is mounted behind the front panel. The DM 163 card may be replaced with an optional DM 156, Power Fail/Restart Card, P/N 44P1094. The latter contains both auto memory disable and power fail/restart circuits.

Expansion Chassis

Expansion chassis used to hold additional memory modules and I/O controllers are identical in size and construction to the mainframe chassis. The blank front panel hinges in the same manner and the dimensions shown in Figure 26-4 apply to both types of chassis.

Each expansion chassis contains its own power supply (controlled by the mainframe on-off switch) and space for four circuit-board trays. The tray spaces can be filled by four 4K-byte memory trays, as shown in Figure 26-6, or by four 8-bit I/O controller

trays, as shown in Figure 26-7, or by any combination of the two types of trays, as shown by the example in Figure 26-8. Each I/O controller tray can accommodate up to four 8-bit I/O controllers.

The 16-bit I/O controllers that connect to the 16-Bit Peripheral Adapter in the mainframe require a special chassis adapter since they have been designed primarily for use with a Varian 620/i computer. The chassis adapter occupies two tray spaces and accommodates two 16-bit controllers. Figure 26-1 illustrates this arrangement; the lowest expansion chassis in the illustration contains two 8-bit I/O trays and one 16-bit chassis adapter.

The design of each expansion chassis is not affected by its contents (i.e., memory modules, I/O controllers, or a combination of the two) except for the signal/power distribution board mounted behind the front panel.

An expansion chassis devoted to I/O controllers is equipped with an 8-bit I/O connector board (16-bit controllers plug into the board for power only). An expansion chassis devoted to memory modules is equipped with an expanded-memory connector board. A combination chassis is equipped with both boards, as shown in Figure 26-9.

The standard Varian 520/i power supply is used in all-memory expansion chassis and most combination chassis. An all-I/O chassis may require a +5V expansion power supply, depending on the nature of the controllers.

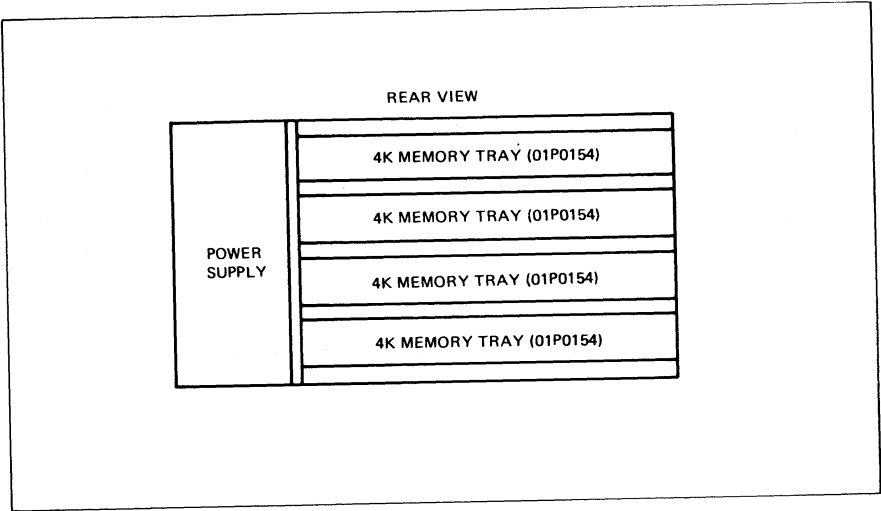


Figure 26-6. Tray Locations, Expansion Chassis with 16K Memory

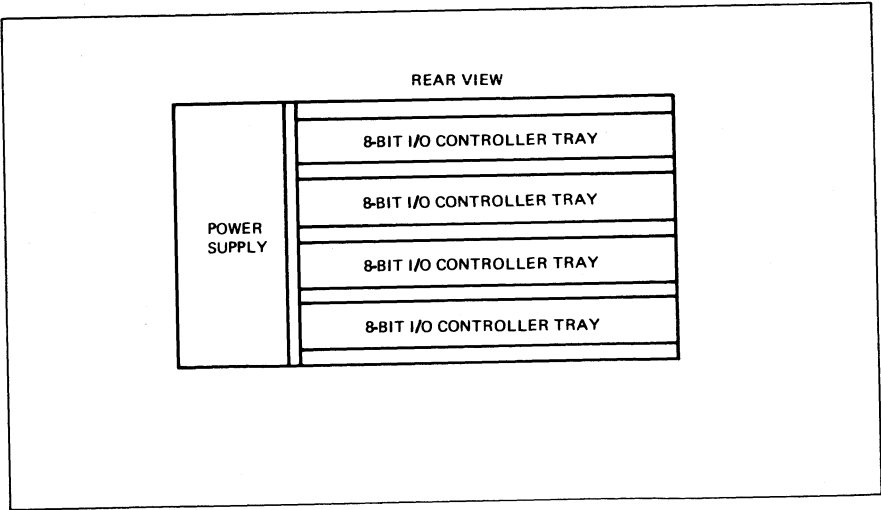


Figure 26-7. Tray Locations, Expansion Chassis with Up to 16 I/O Controllers

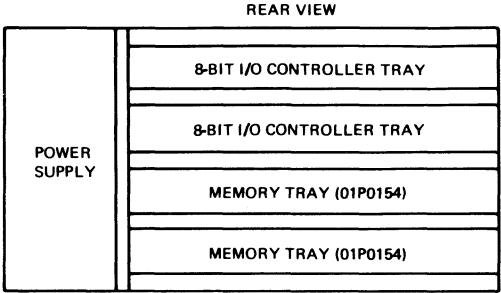


Figure 26-8. Typical Tray Locations, Expansion Chassis with Both Memory and I/O Controllers

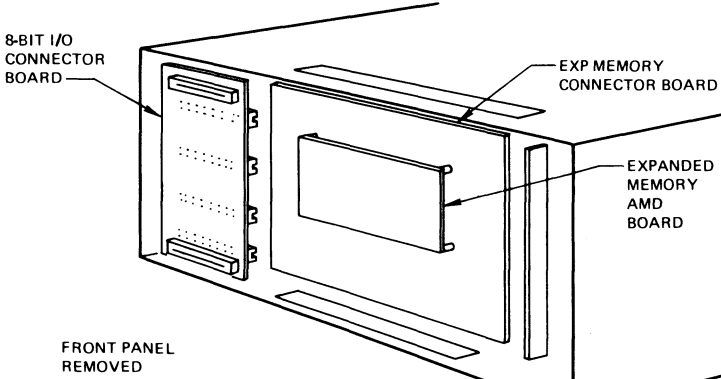


Figure 26-9. Signal/Power Distribution Boards on Expansion Chassis

Memory Expansion and Cabling

The mainframe can house up to 8K-bytes of memory. Additional memory capacity is added in 4K increments up to a total of 32K.

The first four increments are housed in an expansion chassis directly above the mainframe. The final two increments are contained in a second expansion chassis directly above the first.

The block diagram in Figure 26-10 illustrates this arrangement. Each 4K of memory is contained in a tray that plugs into an expanded memory connector board at the front of the expansion chassis. Side and front views of this board are shown in Figure 26-11. The trays plug into connectors J1, J3, J4, and J5. Connectors J2 and J6 are for inter-chassis memory cabling.

The cabling interconnects all memory connector boards on a pin-for-pin basis. The signals carried by the lines are listed in Figure 26-12.

The memory bus from the mainframe must be terminated by a plug-in terminator board, such as that illustrated in Figure 26-13.

Included in the memory cable (Pin 34) is a line connecting the mainframe circuitry to the Auto Memory Disable board mounted on the memory connector board. A SIG TRIG signal from the CPU operates on the X and Y drive circuits in the expansion memory modules exactly as in the mainframe memory modules.

The memory cabling should be installed after the mainframe and memory expansion chassis have been securely bolted in place. One of the following procedures should be used, depending on the total size of the system:

Procedure, 4K or 8K System. No cable included. No action required.

Procedure, 12K through 24K System. Open front panels of both mainframe and first memory expansion chassis and locate memory expansion connectors. Install one end of signal cable into main frame at J2 and route through slot in top of mainframe cabinet. Route cable through slot in bottom of memory expansion chassis and insert into connector J5. Insert terminator into J6 (see Figure 26-11).

Procedure, 28K through 32K System. Open front panels of both memory expansion units and the mainframe. Perform procedure for 24K memory. Insert second memory cable into J6 of first expansion chassis. Route cable up through chassis slots and insert the second end into J5 of the second expansion chassis. Insert terminator into J6 of the second expansion chassis.

8-Bit I/O Cables and Controllers

Each expansion chassis can hold up to four 8-bit I/O trays as illustrated in Figure 26-14. Each I/O tray can hold, in turn, up to four 8-bit I/O controllers.

Expansion chassis designed for I/O service are equipped with an I/O connector board behind the front panel, as illustrated in Figures 26-9 and 26-15. The I/O controller

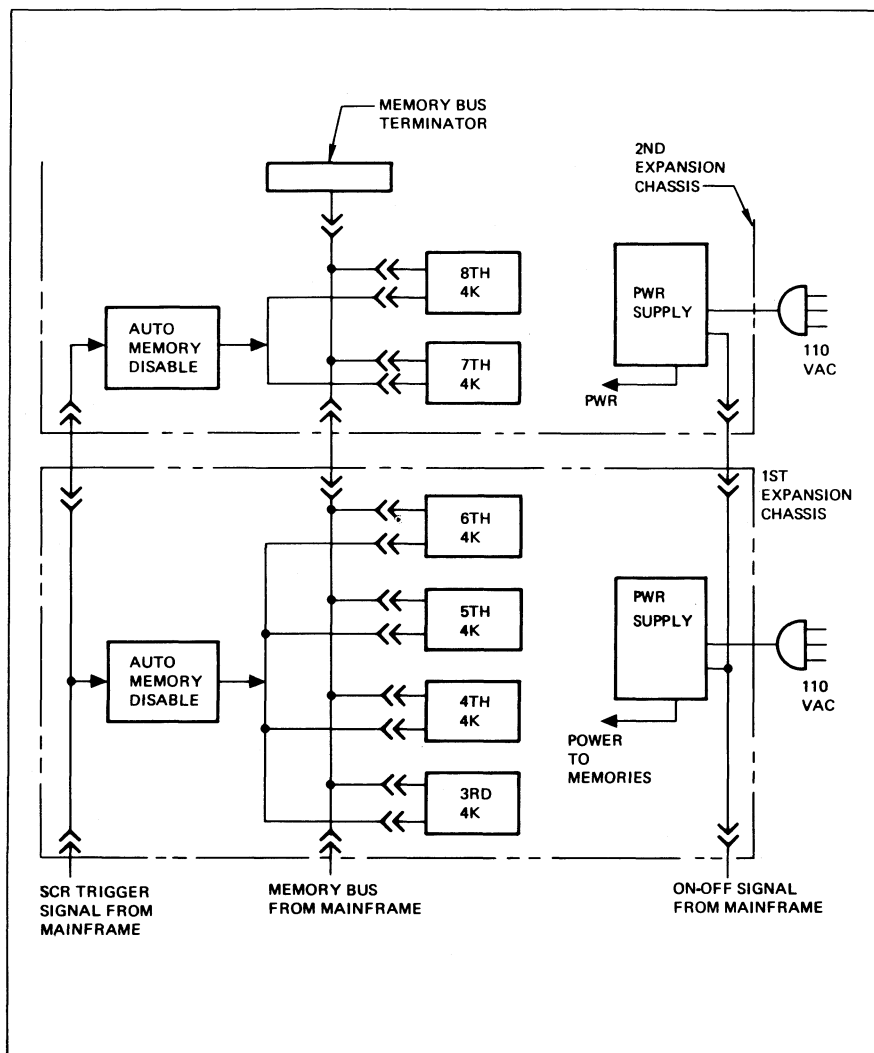


Figure 26-10. Block Diagram of Expanded Memory

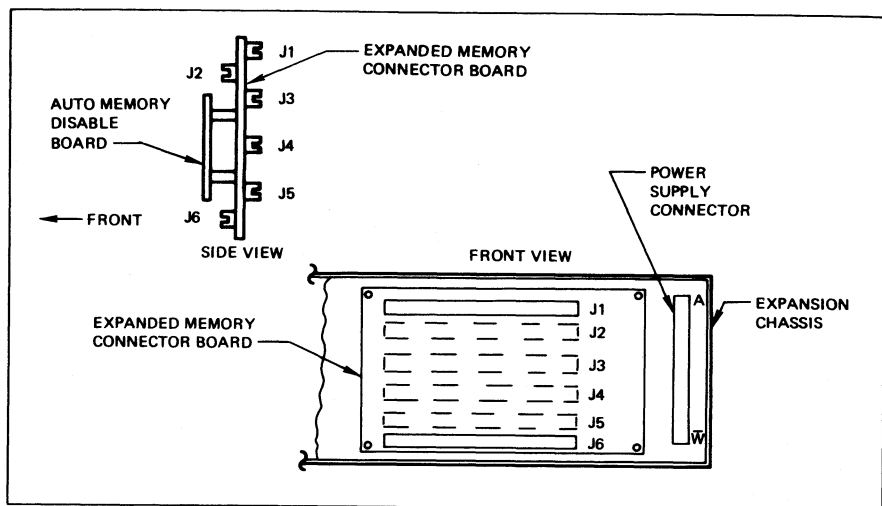


Figure 26-11. Expanded Memory Connector Board

trays plug into connectors J2, J3, J4, and J5. The 8-bit I/O bus cabling plugs into connectors J1 and J6.

The 8-bit I/O bus originates on connectors J2 and J3 at the rear of the I/O Interface Board (DM 155) in the mainframe chassis. The locations of these connectors are shown in Figure 26-16 and the pin assignments are tabulated in Figure 26-17 and 26-18.

The I/O cable that connects the mainframe to the first I/O expansion chassis is formed into a flat cable of twisted pairs and laid along the inside upper surface of the expansion chassis, from rear to front. A sheet-metal bracket holds the cable securely in place. Its terminal connector is plugged into connector J1 of the expansion chassis I/O connector board.

Chassis-to-chassis cabling consists of short cables that connect J6 of one I/O connector board to J1 of the next. The 8-bit I/O cabling is shown in profile in Figure 26-19.

The I/O bus is terminated by a circuit board similar in construction to that illustrated for the memory terminator in Figure 26-13. Lines EA00 through EA07 require such termination.

The terminator connector can also be used as a source for an extended I/O bus to peripheral equipment.

The basic design of the I/O controller tray is illustrated in Figure 26-20. A single circuit board is divided into four sections, one for each of four controller circuits. Mounted above this board is a driver/receiver board

Pin No.	Signal	Pin No.	Signal
1	GND	34	DAG (Memory Enable)
2	GND	35	M READ+
3	GND	36	MEM ADDRESS
4	-12V	37	MEM ADDRESS
5	+12 V	38	MEM ADDRESS
6	V+	39	MEM ADDRESS
7	V-	40	MEM ADDRESS
8	1x+	41	MEM ADDRESS
9	1x-	42	CW+
10	1y+	43	Test Point
11	1y-	44	TMTR-
12	MA00+	45	TMTR+
13	MA01+	46	MDAT00+
14	MA02+	47	MDAT01+
15	MA03+	48	MDAT02+
16	MA04+	49	MDAT03+
17	MA05+	50	MDAT04+
18	MA05+	51	MDAT05+
19	MA07+	52	MDAT06+
20	MA08+	53	MDAT07+
21	MA09+	54	MDAT08+
22	MA10+	55	No Signal
23	MA11+	56	SR00+
24	MB08A+	57	SR01+
25	MB07A+	58	SR02+
26	MB06A+	59	SR03+
27	MB05A+	60	SR04+
28	MB04A+	61	SR05+
29	MB03A+	62	SR06+
30	MB02A+	63	OPEN
31	MB01A+	64	OPEN
32	MB00A+	65	GROUND
33	MWRIT+		

Figure 26-12. Pin Assignments, Memory Connectors

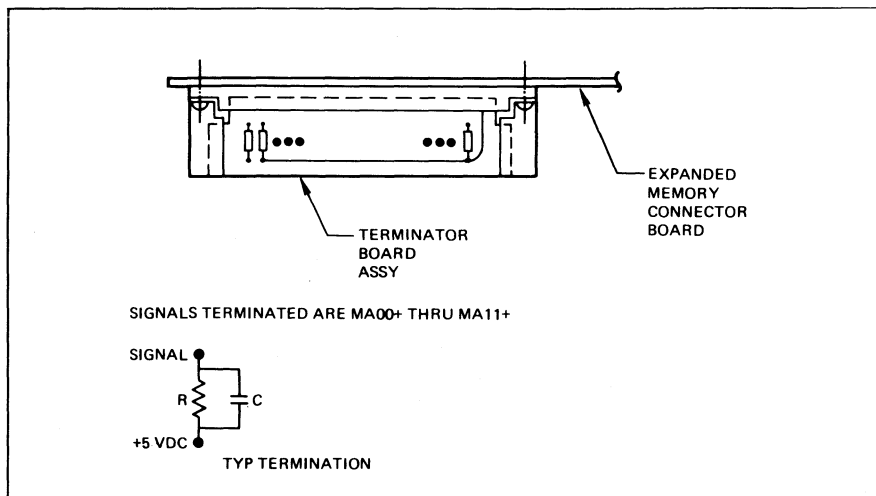


Figure 26-13. Terminator Board for Memory Bus

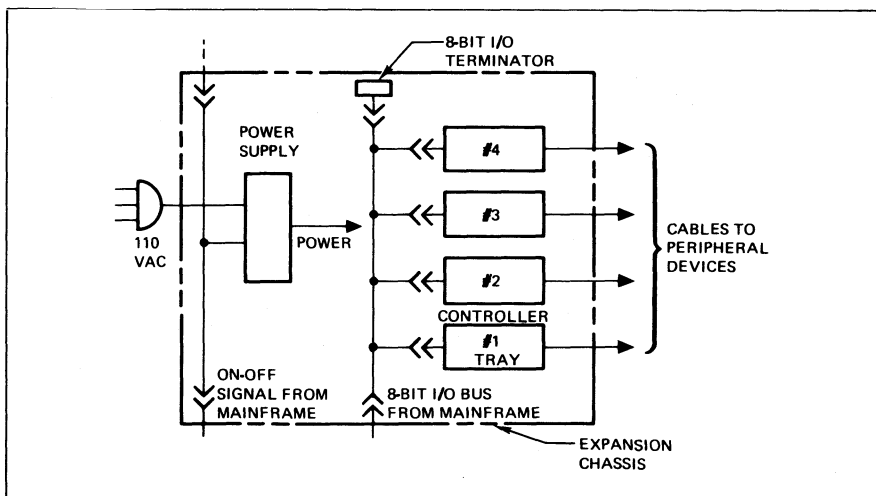


Figure 26-14. Block Diagram, 8-Bit I/O Expansion Chassis

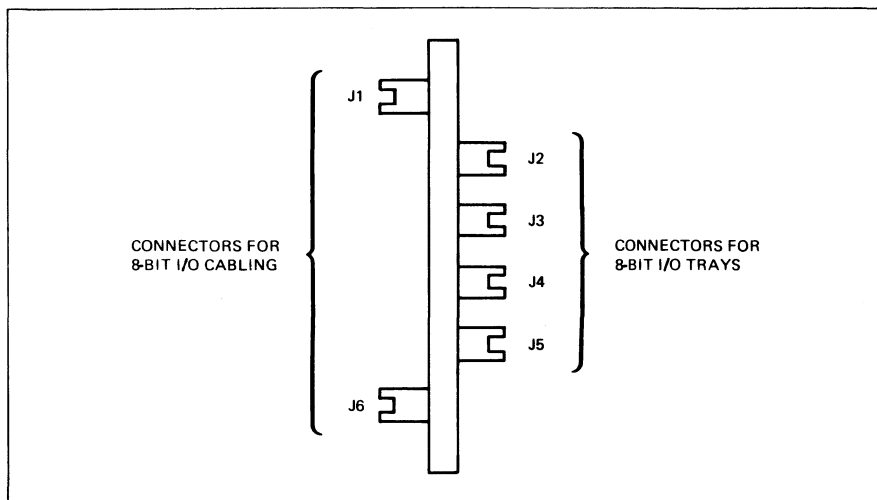


Figure 26-15. I/O Connector Board

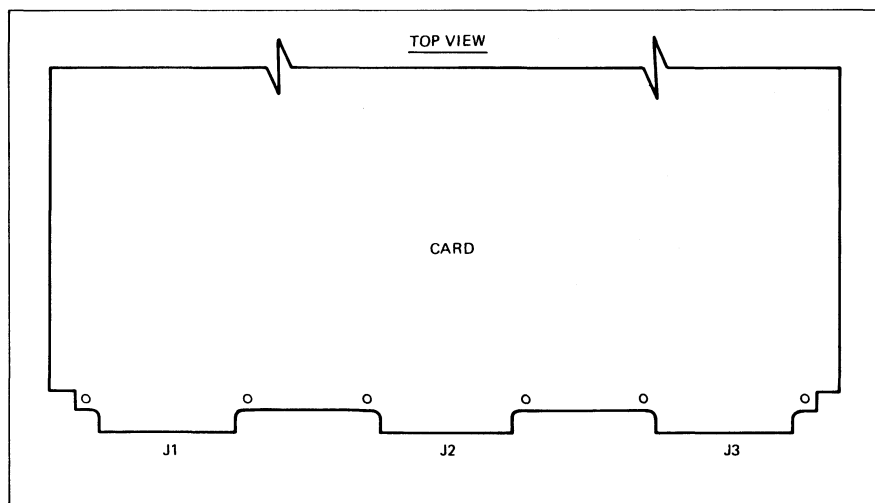


Figure 26-16. Card Connector Locations on I/O Interface Board

PIN	Function
1	EA01- }
3	EA00- }
5	EA03- }
15	EA04- }
17	EA02- }
19	EA07- }
21	EA06- }
23	EA05- }
31	BT0S- }
33	SENS- }
35	BTIS- }
37	CRUN- }
39	RSTS- }
41	FUNS- }
	Address Lines
	Control Lines

Figure 26-17. Pin Assignment, Connector J2, I/O Interface Board

that is shared by the four controller circuits. Figure 26-21 is a block diagram of the relationship between the shared driver/receiver board and the individual controller circuits.

The I/O controller trays are also available in one, two, or three section configurations. These can be expanded later by adding the required output connectors, bolting additional controller circuit cards in place, and installing jumper cables to the driver/receiver board.

16-Bit I/O Cables and Controllers

A peripheral chassis adapter is required to accommodate a 16-bit I/O controller in an I/O expansion chassis. Each chassis adapter

can hold two 16-bit I/O controllers, as illustrated in Figure 26-22.

The 16-bit controllers are designed primarily for the Varian 620/i and can be used in a Varian 520/i system only through the addition of the 16-Bit Peripheral Adapter option (Section 29).

As shown in Figure 26-23, the chassis adapter has a plug, P2, that fits into the 8-bit I/O connector board, but only for dc power. Signal input from the mainframe and signal output to the peripheral device is through connectors at the rear of the chassis adapter.

Two types of cables are required to construct the 16-bit I/O bus, as illustrated in Figure 26-24. One interconnects the 16-Bit

Pin	Function
3	ED07- }
5	ED06- }
7	ED04- }
9	ED05- }
11	ED02- Data Lines
13	ED03- }
17	ED00- }
19	ED01- }
21	SERPS- Sense Return
23	EI01- }
25	EI00- }
27	EI02- }
29	EI09- }
31	EI03- }
33	EI10- Interrupt Lines
35	EI08- }
37	EI08- }
39	EI04- }
41	EI06- }
43	EI05- }

Figure 26-18. Pin Assignment, Connector J3, I/O Interface Board

Peripheral Adapter Board (DM 160) in the mainframe to the first expansion chassis containing a peripheral chassis adapter. The second type of cable connects one expansion chassis to the next.

Signals carried by the 16-bit I/O bus are listed in Section 27, Figure 27-1.

Interrupt Connections

A total of eleven interrupt lines are available for connecting external interrupt signals. These are combined into four priority levels, as discussed in Section 9.

Figure 26-25 summarizes the ways in which these interrupt lines can be used. Interrupts generated by peripheral devices on the 8-bit bus are brought out via connector J3 of the I/O Interface (DM 155) card.

The interrupt line (PTINT) of the paper tape controller is wired to pin 9 of connector J1 on the DM 146 card. (Inputs of connector J3 of the DM 155 card are logically ORed with inputs of connector J1 of the DM 146 card.)

Figure 26-26 lists the 16-Bit Peripheral Adapter interrupts which correspond to the

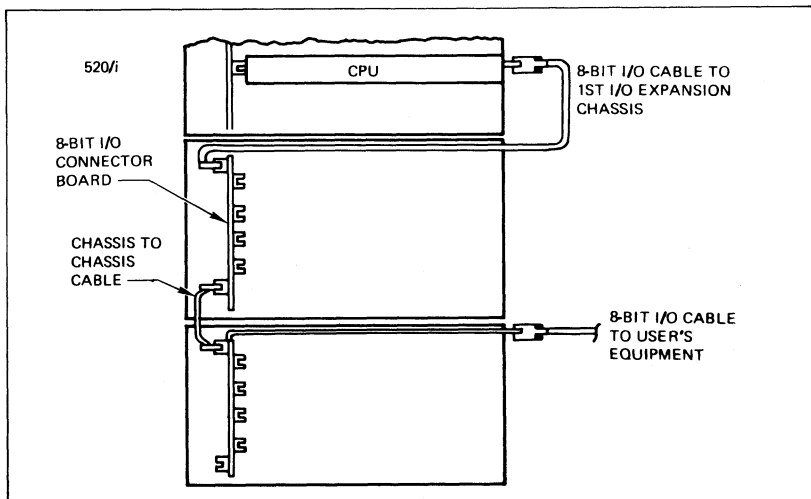


Figure 26-19. 8-Bit I/O Bus

various pins of socket locations Z13 and Z14 on the CPU socket card (DM 141). Internal interrupts can also be accomplished at these locations. An example of this is the teletype interrupt (TINT) shown in Figure 26-25. The teletype interrupt signal, from locations Y9-10 and Y16-11 on the DM 141 card, can be connected to any one of the eleven pins of socket locations Z13 or Z14 listed in Figure 26-26.

It should be noted that TINT comes from two gate circuits that have a common output; a READ interrupt signal is available at location Y16-11 and a WRITE interrupt at location Y9-10. If separate READ and WRITE interrupts are desired, these two points should be separated and jumpered to the interrupt lines as desired. If separate interrupts are used, a 1,000 ohm pull-up

resistor (to +5 volts) must be connected to Y9-10.

For external devices operating on the 8-bit I/O bus, the individual interrupt lines should be traced and the appropriate connections made in the cable between the I/O controller and the device. The pin numbers of connector J3 (DM 155) and the corresponding interrupts are listed in Figure 26-27 for a convenient reference.

Power Supplies

All dc voltages required by the Varian 520/i are supplied by self-contained power supplies housed within each mainframe and expansion chassis.

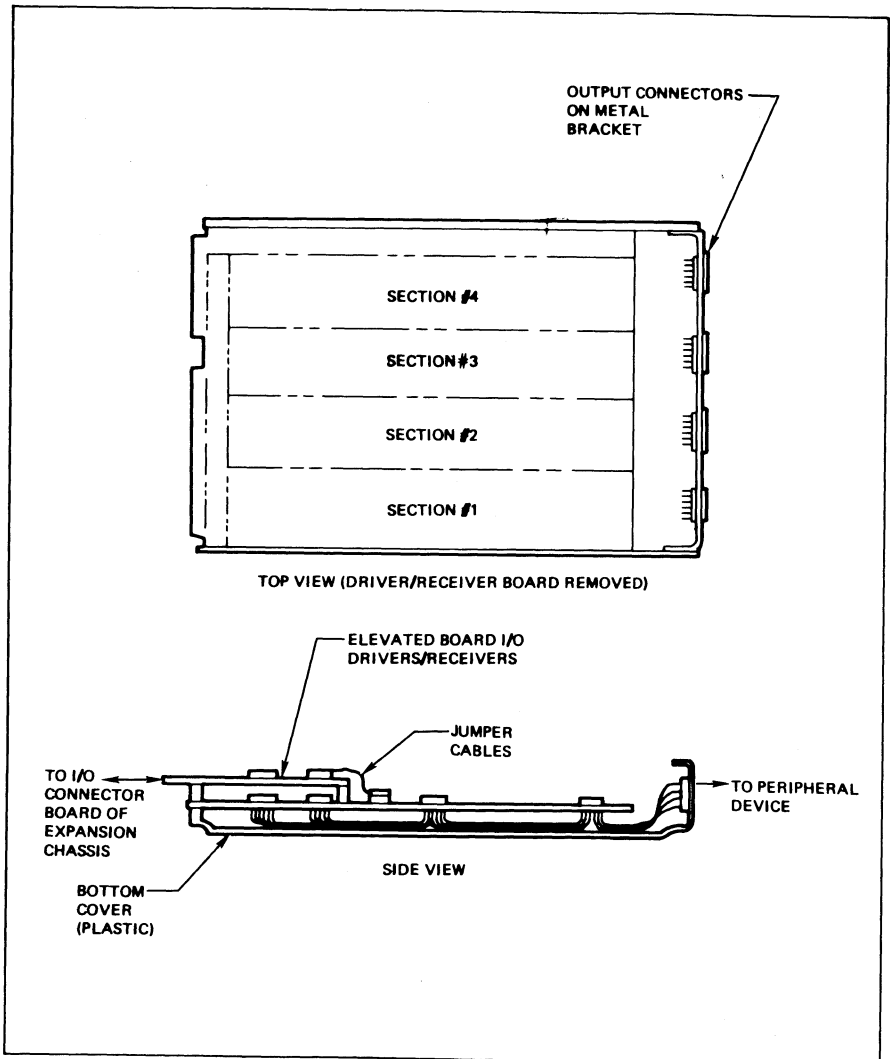


Figure 26-20. 4-Section I/O Controller Tray

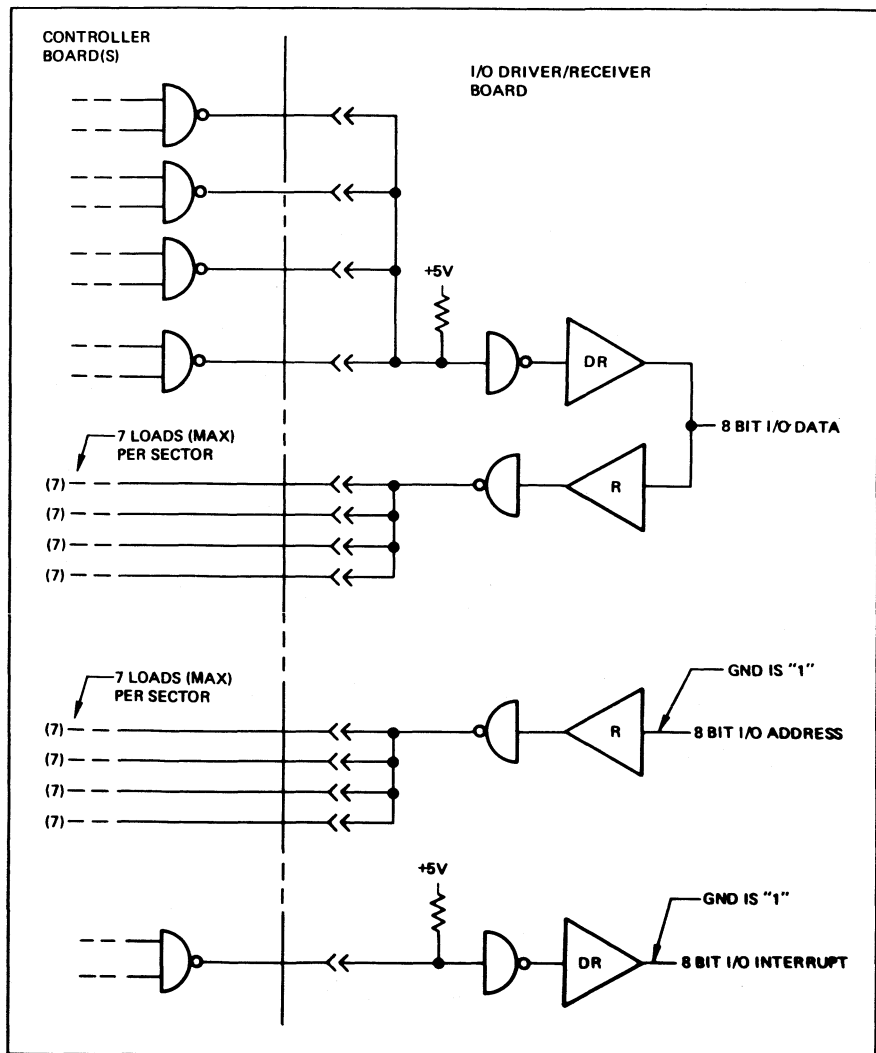


Figure 26-21. Drivers and Receivers, I/O Controller Tray

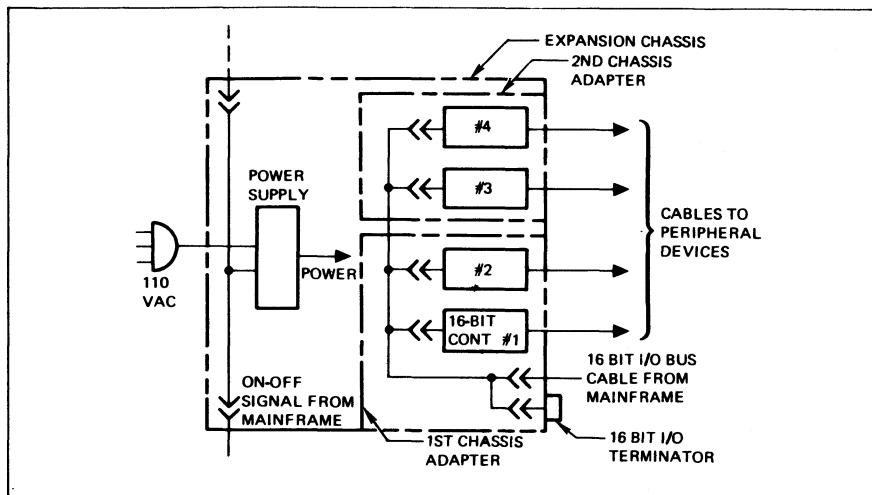


Figure 26-22. Block Diagram, 16-Bit I/O Expansion Chassis

The dc voltages supplied are +5, +12, -12, and a variable voltage of +16 to +23 volts. The three fixed voltages are referenced to ground while the variable output is referenced to -12 volts.

The power supplies are normally wired for 115-Vac operation; however, units wired for 230-Vac operation are also available. Converting the power supply for 230-Vac operation is performed at the factory. It is accomplished by changing jumper wires and replacing the 115 Vac power relay with a 230 Vac power relay.

Figure 26-28 summarizes the power requirements of various Varian 520/i system components. The standard Varian 520/i power supply consists of three supplies

capable of providing 22.0 amperes of +5.0 volts, 0.40 amperes of +12 volts and 0.70 ampere of -12 volts. This is sufficient for the mainframe, for an all-memory expansion chassis, and for most memory-and-I/O expansion chassis. For certain all-I/O expansion chassis, however, the +5 volts supply is not sufficient and an expansion power supply (designated as a +5 V supply) is required.

On-off control of supplies in the expansion chassis is controlled by the "on-off" switch on the front panel of the mainframe. The technique used is shown in Figure 26-29. Note that the wiring of the ac outlets for all supplies in the system must be consistent so that all supplies are in phase with regard to the 110 Vac coming to the relay.

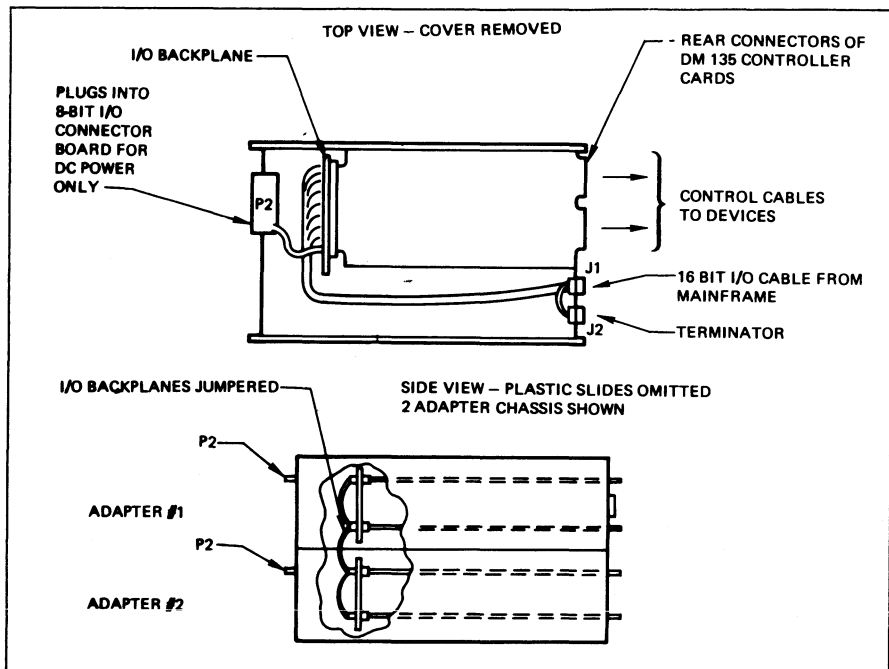


Figure 26-23. 16-Bit Peripheral Chassis Adapter

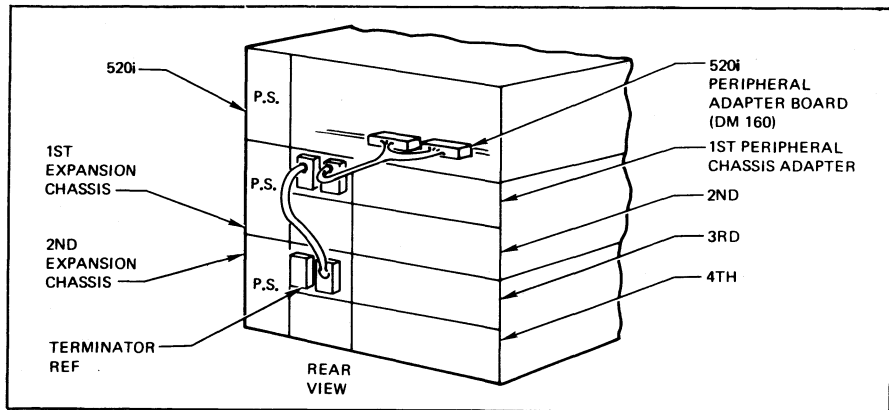


Figure 26-24. 16-Bit I/O Bus Cabling

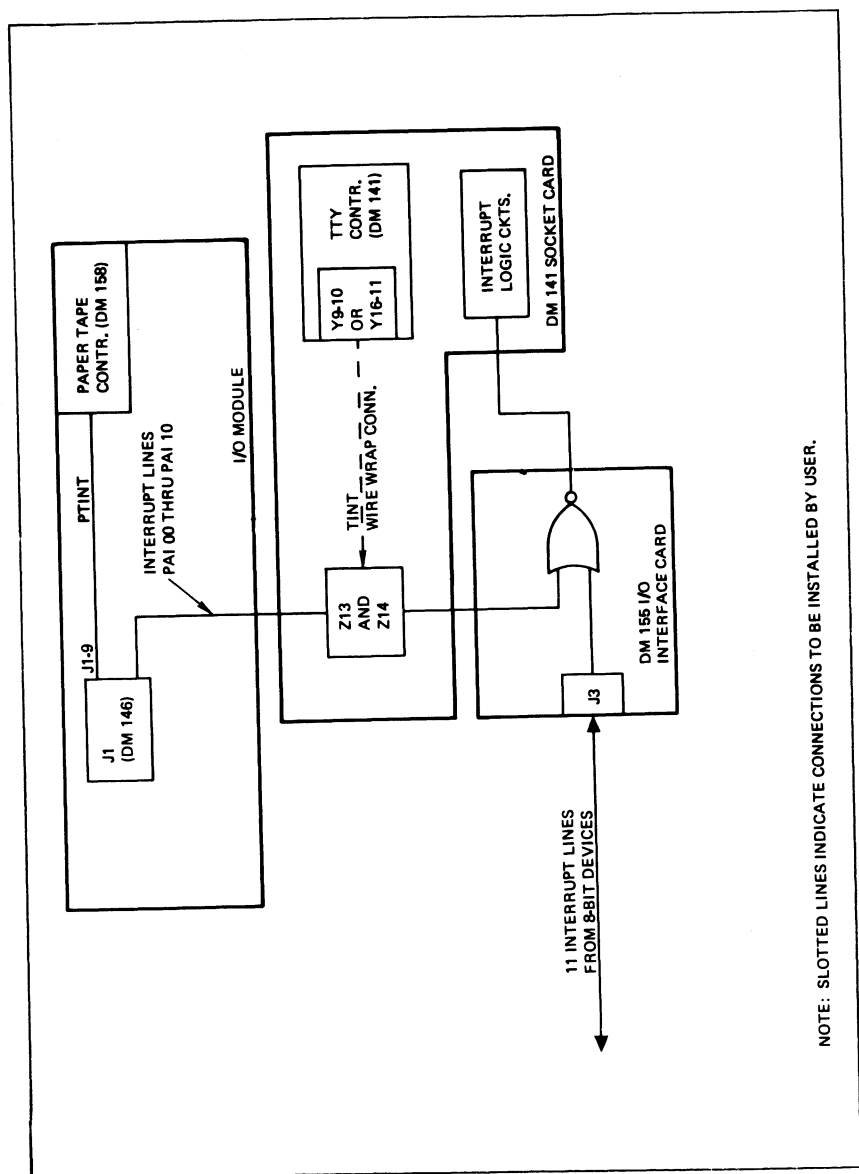


Figure 26-25. Block Diagram, Interrupt Connection

Peripheral Adapter Interrupt Line	Z13 and Z14 Connector Pins
PAI 00	Z13-6
PAI 01	Z13-8
PAI 02	Z13-9
PAI 03	Z13-10
PAI 04	Z13-11
PAI 05	Z13-12
PAI 06	Z13-13
PAI 07	Z14-1
PAI 08	Z14-2
PAI 09	Z14-3
PAI 10	Z14-4

Figure 26-26

External Interrupt Line	J3 Connector Pin
EI00	25
EI01	23
EI02	27
EI03	31
EI04	39
EI05	43
EI06	41
EI07	37
EI08	35
EI09	29
EI10	33

Figure 26-27

Equipment	Amperes at 115 vac, 60 Hz
DATA 520/i with up to 8K of memory	4
Model 33 teletype	3
Model 35 teletype	6
600 cpm card reader	2.6
1100 cpm card reader	15
Disc memory	8
Magnetic tape unit (Ampex)	8
300 lpm line printer	20
Paper tape punch	2
Paper tape reader	2
Paper tape spooler	3
Magnetic tape unit (PEC)	2
Data communication controller:	
Main power supply	5
Expansion power supply	5
Expansion chassis (each)	
Memory only	3
I/O only	5
Memory and I/O	5

Figure 26-28. Power Requirements for Varian 520/i System Components

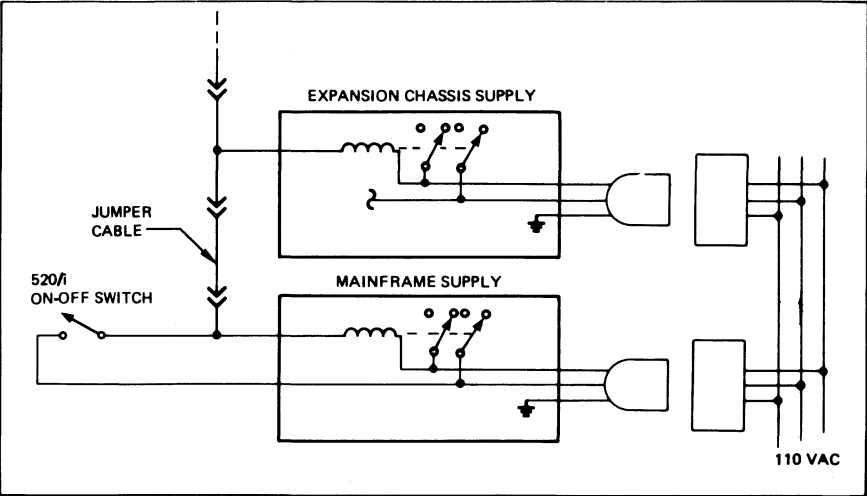


Figure 26-29. On-Off Control and Expansion — Chassis Power Supplies

SECTION 27 — 16-BIT PERIPHERAL ADAPTER

The 16-Bit Peripheral Adapter is an option that can be added to the Varian 520/i computer to provide a 16-bit I/O capability. The additional I/O capability is separate from and in no way affects the operation of the standard 8-bit I/O bus described in Section 8.

The principal function of the option is to provide an interface for the 16-bit peripheral controllers used with the Varian 620/i computer.

The 16-Bit Peripheral Adapter can also serve as a controller interface for any of a number of other peripheral devices that have natural 12-to-16-bit formats (e.g., analog-to-digital converters, card readers, and CRT displays). Folding data from these devices into 8-bit bytes can be both cumbersome and unnecessary; by setting the Varian 520/i precision at two bytes, the computer can deal directly with the data as 16-bit words.

To fulfill its function as an interface for Varian 620/i controllers, the 16-Bit Peripheral Adapter provides an interface that is virtually identical to that of the Varian 620/i I/O system. The only significant differences are the lack of certain interrupt and trap features and a limit of 32 peripheral addresses.

Interface electronics are identical to the Varian 620/i, both in logic levels and drive

capabilities. The controllers for up to 10 peripheral devices may be driven by the Adapter circuitry, as is the case with the Varian 620/i, and timing is fully compatible.

Because the operations are so similar, the output and input connections for the Peripheral Adapter are identified with the same terms as those used for the Varian 620/i. Thus, the Peripheral Adapter I/O lines are identified as the E Bus in the illustrations that follow, and the pin assignments shown in Figure 27-1 carry the same designations used with the Varian 620/i.

EBOO-1 to EB15-1 refer to the 16 address and data lines; FRYX-1 is the function-control line; SERX-1 is the sense-control line; and DYRX-1 signifies data-transfer control. The latter are directly equivalent to the Varian 520/i FUNS-, SENS-, and BT1-/BTO- signals and are generated by them. An additional signal, SYRT-1 (System Reset), is generated in the Peripheral Adapter by the initialize signal from the CPU and is available as a control signal for peripheral devices.

Varian 620/i Controllers

Figure 27-2 lists the standard Varian 620/i controllers that can be connected directly to the Varian 520/i through the 16-Bit Peripheral Adapter. Hexadecimal Varian 520/i device addresses are shown, along with the octal address that would be used if the

16-bit peripheral adapter

75 PIN CONNECTOR – CONNECTOR PIN ASSIGNMENTS

Pin	Function	To
1	EB00-1	
2	R	
3	EB01-1	
4	R	
5	EB02-1	
7	R	
8	EB03-1	
10	R	
11	EB09-1	
12	R	
13	EB05-1	
14	R	
15	EB06-1	
16	R	
17	EB07-1	
18	R	
20	EB08-1	
21	R	
22	EB09-1	
23	R	
24	EB10-1	
25	R	
26	EB11-J	
27	R	
28	EB12-1	
29	R	
30	EB13-1	
31	R	
32	EB14-1	

Pin	Function	To
33	R	
34	EB15-1	
35	R	
36		
37	R	
38		
39	R	
40	FRYX-I	
41	R	
42	DRYX-I	
43	R	
44	SERX-I	
45	R	
46		
47		
48		
49		
50		
51		
52		
53		
54		
55		
56		
57		
58	SYRT-I	
59	R	
60	Open	
62	R	

Pin	Function	To
63		
64		
65		
66		
67		
70		
71		
72		
73		
74		
75		
76		
77		
78		
79		
80	+3VDC	
82	GND	

NOTES:

Figure 27-1. Pin Assignments, I/O (E-Bus) Cable Connector, 16-Bit Peripheral Adapter

controller were part of a Varian 620/i system.

The controllers operate identically with both types of computers, with the following limitations in the case of the Varian 520/i:

- The magnetic-tape controllers cannot be operated in the shared mode.
- The ADC and Buffered I/O controllers require an address modification.

The paper-tape controllers listed in Figure 27-2 should not be confused with the Varian 520/i paper-tape system described in Section 13. Complete information on the listed controllers can be found in the appropriate Varian 620/i handbooks and manuals.

Functional Description

The relationship between the CPU and the 8-bit and 16-bit I/O channels is shown in Figure 27-3. The 8-bit data lines transfer data to and from the least significant 8 bits of the current accumulator (A Register). The 16-bit data lines transfer data to and from a 16-bit buffer that is part of the Peripheral Adapter. This buffer exchanges data, in turn, with the first and second bytes of the accumulator.

The Peripheral Adapter also compensates for the fact that in the case of the Varian 520/i 8-bit I/O bus, data and peripheral-device address are carried simultaneously by two separate sets of lines (see Section 8 for details), whereas the Varian 620/i uses a single set of 16 lines to carry first the peripheral-device address, then the data.

The block diagram for the 16-Bit Peripheral Adapter, Figure 27-4, indicates how the option performs these functions. The 16-bit buffer mentioned above is shown as the F Register, divided into two 8-bit sectors. The F Register is treated by the CPU as an operational register and data can be transferred between it and any CPU register by using the standard set of register-transfer instructions. The gating into the upper or lower sectors is determined by the timing and control logic.

The Varian 620/i control signals, FRYX- and DRYX- are generated by the timing and control section in response to a BTI, BTO, SEN or FUN instruction. The first phase in the execution of such an instruction is to transmit a 16-bit word on the E-Bus containing the address of the peripheral device to be activated, a function code, and a 4-bit field indicating whether a FUN, SEN, BTI, or BTO operation has been initiated. The format of this address-and-control word is shown in Figure 27-5.

Following the transmission of this address and control word, in the case of BTI and BTO operations, a 16-bit data word is placed on the 16-bit bus. The data transfer can be either an output, from F Register to controller, or an input, from controller to F Register. The F Register is not involved in SEN or FUN operations.

As shown in Figure 27-4, the address and function-code portion of the initial address-and-control word are derived directly from the Varian 520/i memory data bus. The information on the memory data bus, indicated in the lower half of Figure 27-5, is, in effect, the second byte of a two-byte Varian 520/i I/O instruction. (In the case of an 8-bit I/O operation, this

16-bit peripheral adapter

Model No.	Device Controller Name	Device Address	
		620/i DA (Octal)	520/i DA (Hexadecimal)
620/i-30	Magnetic Tape Controller 7-Track (with 620/i-30A, or 620/i-30B dual density) May be used with 620/i-30C at 556 bpi only.	10-13	A-D
620/i-31	Magnetic Tape Controller 9-track (with 620/i-31A or 620/i-31B only).	10-13	A-D
620/i-40	Rotating Memory and Controller	14	E
620/i-40	32,768 words, 30 KHz		
620/i-41	65,536 words, 30 KHz		
620/i-42	131,072 words, 30 KHz	32	1A
620/i-43	262,144 words, 30 KHz		
620/i-75	CalComp 565 Plotter		
620-23	Card Reader Controller	30	18
620/i-50	Paper Tape Punch 60 cps Controller	35	1D
620/i-51	Paper Tape Reader 300 cps Controller	37	1F
620/i-52	Paper Tape System and Controller	60-61	30-31
620/i-80	Buffered I/O with Sense & EXC Lines		
620/i-85	Analog-to-Digital Converter Controller		
		54-57	2C-2F

Figure 27-2. Device Addresses for Varian 620/i Controllers

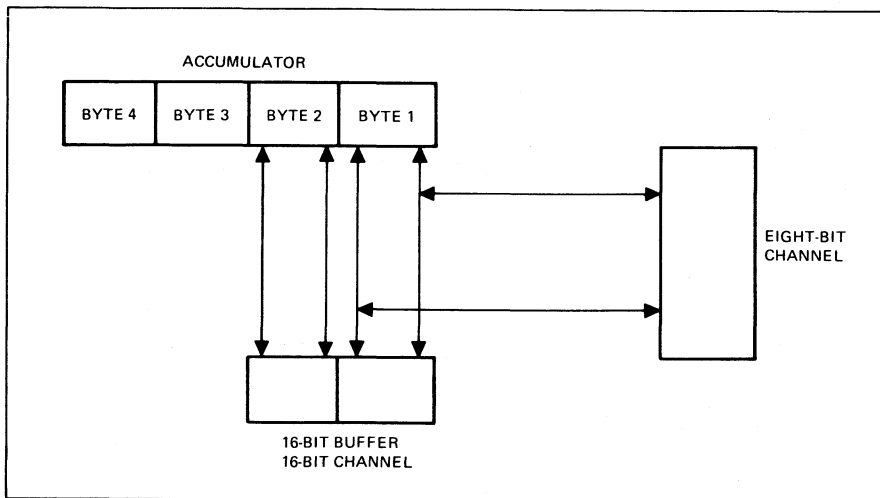


Figure 27-3. Relationship Between 8-Bit and 16-Bit Channels

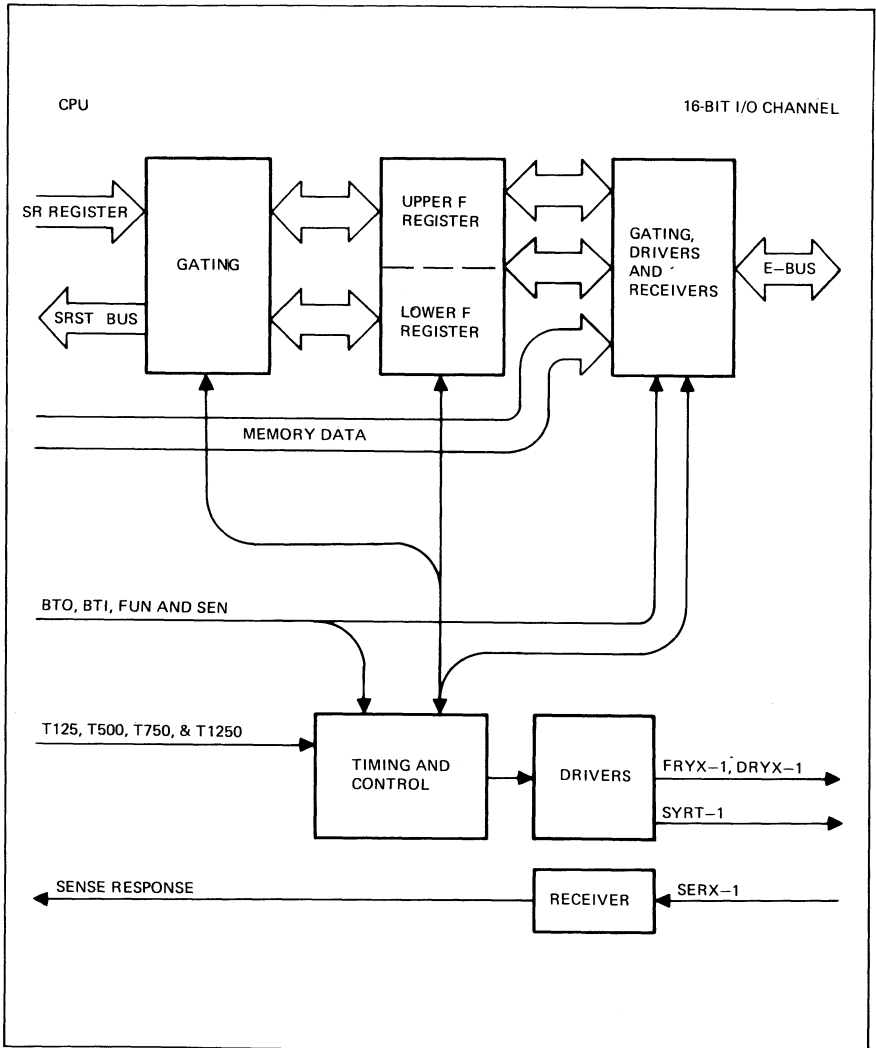


Figure 27-4. 16-Bit Peripheral Adapter Block Diagram

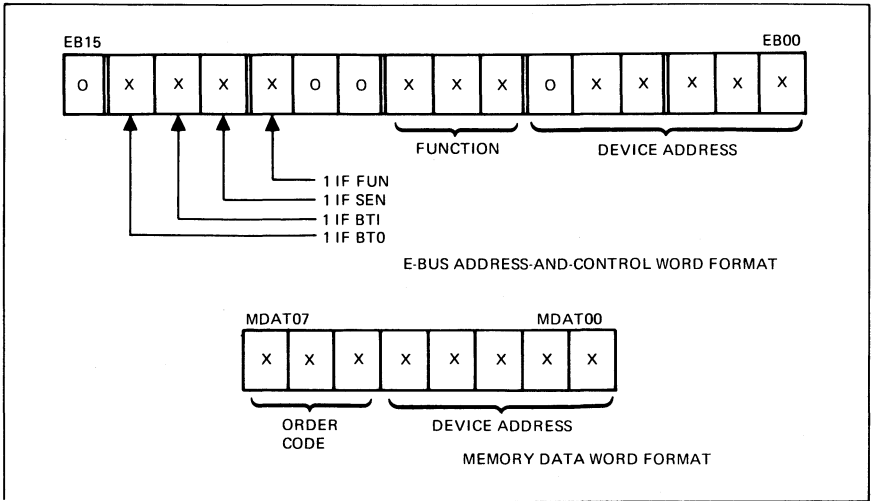


Figure 27-5

second byte is transferred directly from the memory data bus to the 8-bit I/O address bus.)

Since the E-Bus device address is derived from the 5-bit address field of the Varian 520/i I/O instruction, it too is limited to 5 bits, or a total of 32 device addresses. The sixth bit, which in a Varian 620/i system, allows up to 64 devices to be addressed, is always a zero. In translating from I/O instruction to E-Bus format, the 3-bit order code is shifted up one place to make room for the zero in the sixth bit position.

F Register

As noted above, the F Register can be loaded or interrogated by software like any

other 16-bit operational register. However, only transfer operations are recommended.

Incoming 8-bit bytes from the SR Register (see Section 3, Figure 3-1) are placed alternately in either the upper or lower portions of the F Register and only after a word is assembled can it be recognized as the equivalent of a Varian 620/i data word. Like the Varian 520/i, the Varian 620/i processes data in two's complement form, with bit 15 serving as the sign bit.

The F Register can be read or altered at any time except during the execution of a BTI. It is presumed that the F Register is loaded prior to a BTO operation and interrogated after a BTI operation.

During a BTO, the Peripheral Adapter gates the F Register onto the E-Bus at the appropriate time. Similarly, the F Register is loaded by the E-Bus during a BTI under Peripheral Adapter control. The F Register plays no part in a SEN or FUN operation.

Timing

The 16-Bit Peripheral Adapter has two different timing cycles. In the case of a FUN or SEN operation, the cycle is nominally 1,000 nanoseconds. A BTO or BTI operation requires a nominal 2,750 nanoseconds.

All signals generated in the Peripheral Adapter are based on the 4 MHz clock rate of the Varian 620/i. Thus (neglecting gate delays) all signals have durations that are multiples of 250 nanoseconds. To be compatible with Varian 620/i controllers, however, the signals must also correspond with those on the Varian 620/i E-bus.

Figure 27-6 compares the signals generated by the Peripheral Adapter and by the Varian 620/i. Following Varian 620/i convention, the T_0 reference in both cases is the appearance of the bit indicating that a FUN, SEN, BTI, or BTO operation is underway. The bit is a one in each case and is the first activity on the E-bus seen by the controller.

The waveforms of Figure 27-6 represent all four operations. Only the first phase (T_0 to T_{1130}) apply to the FUN or SEN operations. The D_{YRX}- signal has two different durations, depending on whether a BTO or BTI operation is being performed.

The dashed lines on the Peripheral Adapter waveforms represent worst case gate and flip-flop delay accumulations. These worst-

case situations are referenced to the worst-case early and late data on the E-bus. Thus, the early E-bus rise time of 885 nanoseconds is referenced to the latest data on the E-bus. It cannot be compared with the late FRYX-rise time of 815 nanoseconds, which is referenced to the earliest data on the E-bus. As a result, the Peripheral Adapter waveforms should not be compared with each other, but only with corresponding waveforms for the Varian 620/i.

It can be seen that the Peripheral Adapter waveforms are slightly later and have longer durations than the Varian 620/i waveforms. This will not stress the speed of any device presently capable of keeping up with the Varian 620/i.

The actual phase relationships of the Peripheral Adapter waveforms is shown in Figure 27-6. Here, T_0 is referenced to the basic Varian 620/i clock.

During the first phase of an operation, the Peripheral Adapter gates the address and command code from memory data onto bits 00-08 of the E-bus (bit 5 is a hardwired zero in this mode). Bit 11, 12, 13 or 14 of the E-bus is set to one depending on the operation. FRYX- is driven to ground and returns to -5 before the data is removed.

This sequence completes a FUN or SEN operation. For BTO and BTI operations, data is transferred in the second phase. At T_{1750} , data is gated from the F Register to the E-bus for a BTO. The transfer is reversed for a BTI operation.

During a BTI, the device queried must send data for a minimum of 650 nanoseconds. The data cannot be removed sooner than 150 nanoseconds after the leading edge of

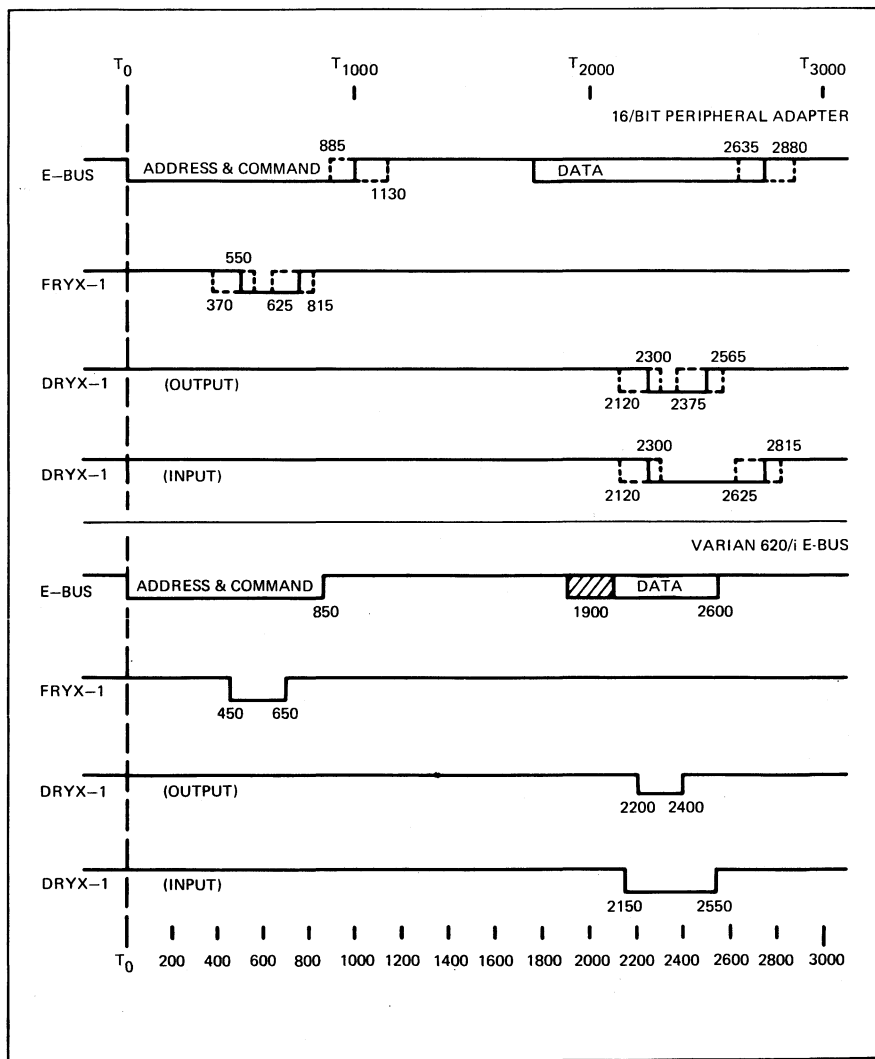


Figure 27-6. Comparison of Peripheral Adapter and Varian 620/i Timing

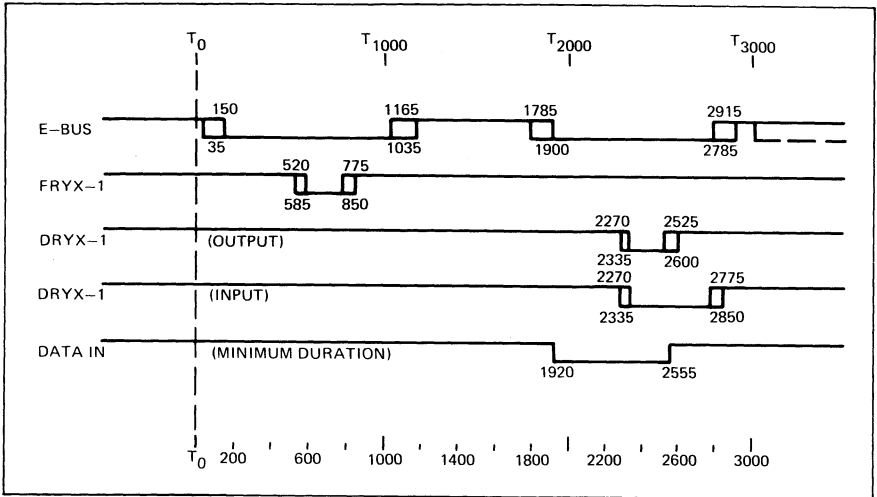


Figure 27-7. 16-Bit Peripheral Adapter Waveforms (Referenced to Varian 520/i Clock)

DRYX-1. Data should be applied no later than 1,000 nanoseconds after the trailing edge of FRYX-1.

Sense response should be applied within 435 ns of the last data bit received and should be up for a minimum of 715 ns and a maximum of 2 microseconds.

Physical Construction

The 16-Bit Peripheral Adapter is mounted on the I/O tray within the mainframe of the computer. Controllers connecting to the 16-bit I/O bus are contained within Expansion Chassis that are rack-mounted below the computer.

Cable connections for the 16-Bit I/O bus are described in detail in Section 26, Integration and Installation.

SECTION 28 — MEMORY PARITY OPTION

The Memory Parity Option provides a method for continuously checking on the operation of the Varian 520/i memory.

During the Write cycle of memory (see Section 4), a 9th bit is generated which indicates whether the data contains an even or odd number of ones. When the byte is read out of memory, the data portion (8 bits) is checked against the parity bit. This provides an indication of whether or not the byte is in error. If an error is detected, the parity option generates an interrupt. The parity interrupt which is in addition to the 11 basic machine interrupts described in Section 9, has priority over all others except power fail.

The parity option operates on the basis of "even" parity. This means that if the data byte (8 bits) contains an odd number of ones, the parity bit will be one, e.g., a given memory location (9 bits) will always have an even number of ones.

The block diagram of the Memory Parity Option is shown in Figure 28-1. Data is selected from data-to-the-memory or data-from-the-memory by the Clear-Write mode. During Clear-Write, parity is generated and sent to bit 08 of the memory.

During a memory-read cycle, memory data is selected and parity is generated. The result is exclusive OR'ed with the bit-08 signal from memory. If the signals are

different, the error flip-flop is set and remains set until the CPU loads the parity interrupt address. If a power-fail interrupt is not in process, the memory address of the next instruction is set to \$0004.

Two instructions apply to the Memory Parity Option:

EPI	FUN 40	\$3340	ENABLE PARITY INTERRUPT
DPI	FUN 60	\$3360	DISABLE PARITY INTERRUPT

These control the state of the parity mask flip-flop. If the mask is set, the parity interrupt is inhibited: if reset, the interrupt is enabled. The instruction IBE (Interrupt Block Enable) must also have been executed to enable a Memory Parity interrupt.

The "even" parity feature automatically detects an unplugged memory or an erroneous address directed to a 4K stack not included in the computer.

Parity Error Subroutine

If a parity error occurs in the execution of a BRU instruction, the record of the location

memory parity option

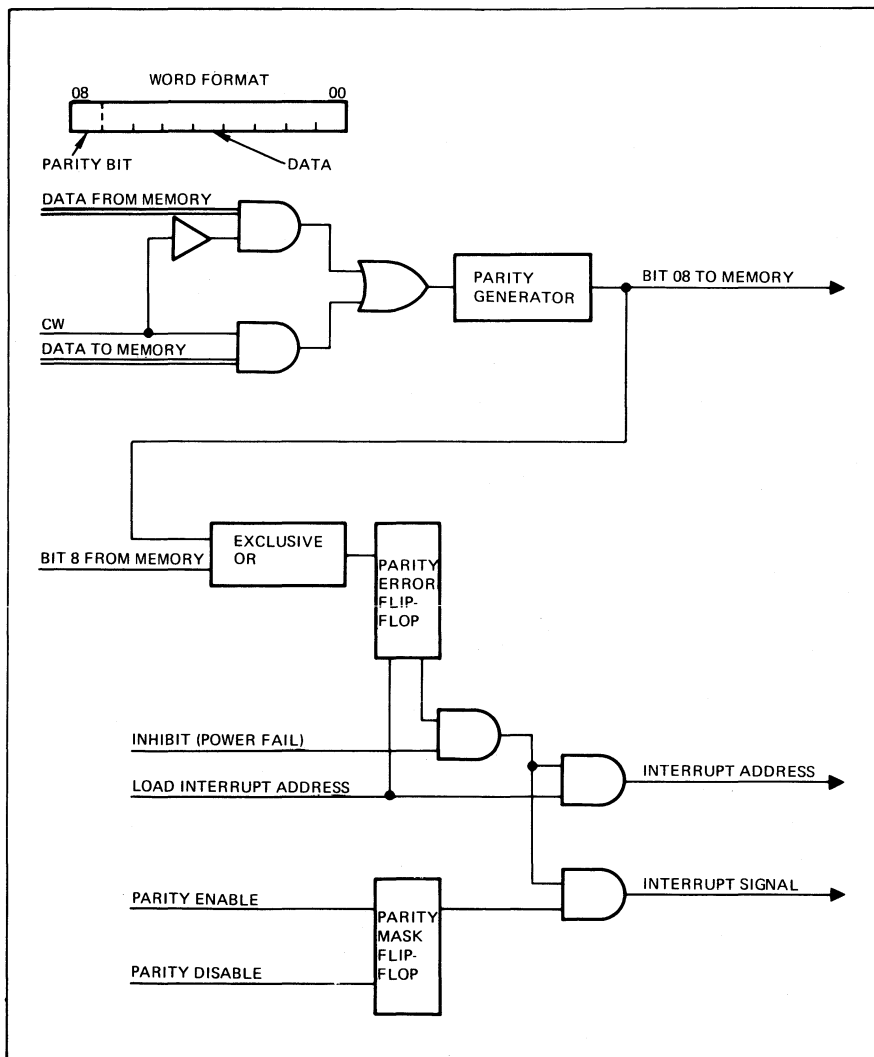


Figure 28-1. Memory Parity Option Block Diagram

of the erroneous byte is lost. The following program is therefore suggested:

04,	SP1	SET PRECISION 1
05,06	TZX	TRANSFER 0 TO X
07,08	TZC	TRANSFER 0 TO LOWER ACCUMULATOR
09,0A	STO-04	STORE ACCUMULATOR (0=HALT) IN LOCATION 04
0B,0C	LODX-00	LOAD INDEXED STARTING AT 00
0D	IXO	INCREMENT INDEX REG'.
0E,0F	BRU-0B	BRANCH TO 0B

Parity Diagnostic Routine

The Parity Diagnostic Routine tests the Memory Parity Option under varying conditions.

Data and instructions are stored in memory with incorrect parity. The data is then read, the instructions executed and a parity interrupt is monitored. If no interrupt occurs, an error message is generated.

The Parity Diagnostic Routine is designed to operate with a Varian 520/i with 4K memory, Memory Parity Option, a Model 33TC ASR Teletype, and a Memory Test Fixture. The location of the Diagnostic Routine is shown in the memory map, Figure 28-2.

The Memory Test Fixture consists of a panel with a Select switch, a Write switch,

and a cable. One end of the cable is inserted into the parity circuit board and the other end connects to a socket on the central-processor board. The function of the fixture is to interrupt the bit (parity bit) signal from the parity generator to the memory. The Select switch may be set either to Normal or Select Parity operation. The Write switch, functional only in the manual mode, controls the parity bit as it is written into memory; either a 1 or a 0 may be selected.

After entry into the Parity Diagnostic Routine, the program halts to allow the hardware to be modified by means of the Memory Parity Test Fixture. The Fixture is set so that data and instructions normally having parity bit set to a 1 are stored with a parity bit of 0. After loading, the Program halts again to allow hardware to be modified to store with a parity bit of 1, data and instructions normally having a parity bit of 0. After the entire table has been stored with "incorrect parity," the program halts to allow hardware to be restored to normal condition.

The computer is then set to its interruptable environment for the initial test. Each byte of data is read and if a parity interrupt occurs, the program branches to the Parity Error interrupt service routine, which sets the correct environment and tests the parity O.K. flag. If the flag is not set, the computer halts. If the flag is set the program restores the data byte and returns to test the next byte.

If no Parity Interrupt occurs, the program branches to an error routine which sets the

memory parity option

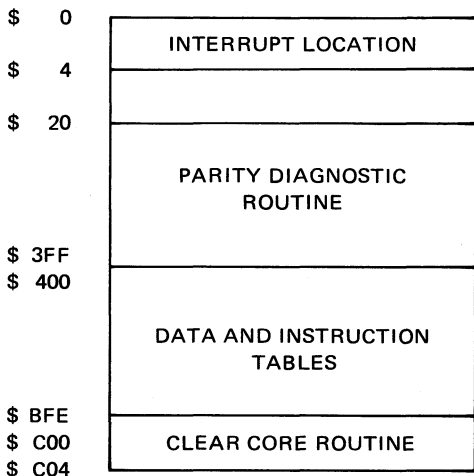


Figure 28-2. Memory Map, Parity Diagnostic Routine

correct environment and outputs an error message with the following format:

XXXX XX * XX XX XX

the six fields representing:

1. Location of bad data
2. Data or instruction
3. Error message load flag
4. Test phase indicator
5. Position of byte in word
6. Current environment indicator

An asterisk in the third field indicates an interrupt on the test load; a blank indicates

an interrupt on the test load but not on the error message load.

Following the error-message output, the program returns to test the next byte.

After all data and instructions have been loaded and instructions have been executed, the test is repeated in the non-interruptable environment.

After both phases of the test have been run, a Restart Test switch may be set (see following paragraphs), and the program halts. If the Restart switch is set, the program returns to retest previously stored incorrect data. If the Restart switch is not set, all of core is cleared of bad data and the program returns to its start location.

The following options are available through use of the sense switches:

Set Sense Switch 1 to halt after error printout

Set Sense Switch 2 to retest same value

Set Sense Switch 3 to suppress printout on error

Operating the Parity Diagnostic Routine

The Parity Diagnostic Routine is operated by the following sequence of actions:

1. Load the parity Diagnostic Routine paper tape into core using the Binary Loader (see Section 15).
2. Set Sense Switches as desired. No other parameters are required.
3. Set the Program Counter to \$002C. Set Processor in RUN mode.
4. When program halts with P = \$002C for machine modification, set SELECT switch on Memory Parity Test Fixture to SELECT PARITY position and WRITE switch to WRITE 0 position. Restart processor. Program stores, with 0 parity, table, test data and instructions having a normal parity of 1.
5. When program again halts with P = \$002C, set WRITE switch to WRITE 1 position and restart processor. Program stores with 1 parity, table, test data and instructions having normal parity of 0.
6. When program halts for machine restoration with P = \$005A, set SELECT switch to EVEN PARITY position and restart processor. Loading and execution of data and instructions takes place.
7. After all data and instructions have been tested in both interruptible and non-interruptible environments, program halts with P = \$00FA. If C Register is not zero, entire test will be run again. If C Register is set = 0, all of core will be cleared.

Error Halts

An error condition exists when either no interrupt occurs with a preset incorrect parity or when a normal parity interrupt occurs.

The program halts with P = \$0181 if a Parity Error is detected and the Error OK Flag is not set. On restart, the bad byte is restored and the test continues.

If Sense Switch 1 is set, the program halts after each error printout. On restart, the program branches to next test.

SECTION 29 — POWER FAIL/RESTART ROUTINE

The purpose of the Power Fail/Restart Routine is to protect an operating program in the event of a power failure by saving the contents of all volatile registers, the state of the system including environment, mode and precision, and to resume operation of the program in its original state when power is restored.

One unique feature of the Power Fail/Restart Routine is the ability to recognize a power down condition while in the step mode; and to return the system to a like state when power is restored.

The Routine is supplied as part of the Power Fail/Restart Option. Other hardware requirements are a Varian 520/i computer with 4K of memory and a Model 33TC Teletype.

Upon detection of a power down condition, the routine has 250 microseconds to store the contents of all volatile registers. The Store routine as implemented requires 174-198 microseconds (see Figure 29-1).

On return of power there is an 800-ms delay before a power up interrupt is generated to the Restore subroutine. The Restore routine as implemented requires 115-124 microseconds.

Functional Description

The Power Fail/Restart Routine consists of two subroutines, both initiated by the same interrupt (see Figure 29-2).

When a power-failure interrupt is generated, control is transferred to the Save subroutine. The 4-byte accumulator is saved to be stored in the proper precision. The Restore subroutine address is stored in the Interrupt location which is common for both routines (0-3). Environment and overflow conditions are determined and appropriate flags are set. The Set Precision instruction is constructed from the precision on entry, and the accumulator is stored accordingly. The remaining registers are stored. The Program Counter is updated, the status changed to the other environment, and all of the above steps are repeated for the new environment.

Step or run mode is determined by sense command. If sense is true, system was in the step mode and will halt at end of the restore routine. If sense is false, control will be transferred back to the operating program on completion of the restore routine.

When a power-up interrupt is generated, control is transferred to the Restore sub-

power fail/restart routine

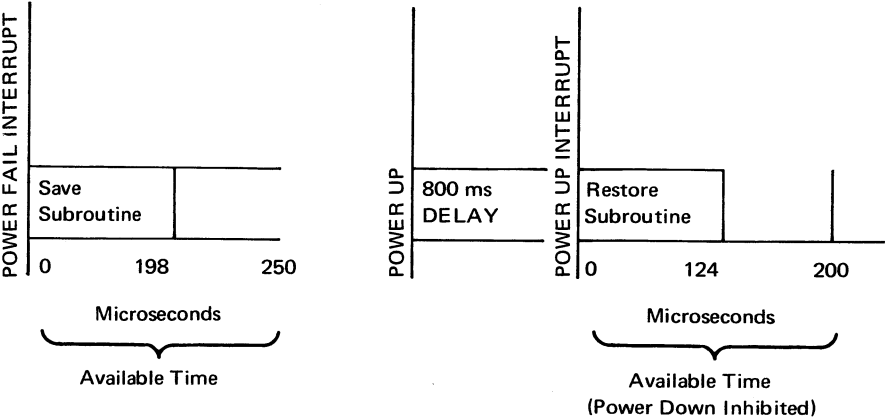
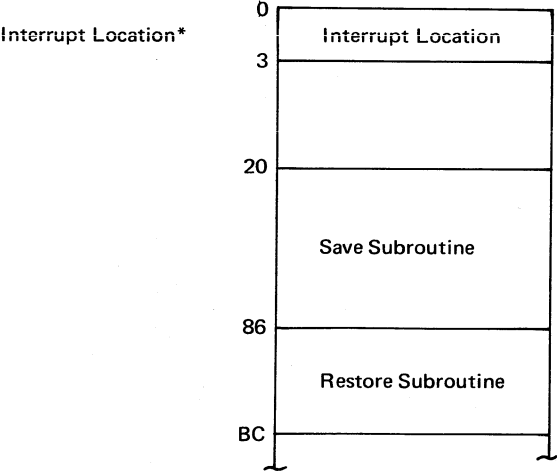


Figure 29-1. Timing Diagrams, Power Fail/Restart Routine



*Same location for both power fail and restart

Figure 29-2. Memory Map, Power Fail/Restart Routine

routine. The assumed conditions on entry are: interruptible environment, precision 1, and no overflow. During the Restore routine, the non-current environment conditions are restored first, then the current environment conditions, leaving the system in the current environment at the end of restore routine.

The address of the Save subroutine is restored to the interrupt location. If the system was in run mode when power failed, control is returned through the entry point of the Save subroutine to the operating program. If the system was in step mode, the routine halts. After restart, control is transferred to the operating program.

SECTION 30 — REAL-TIME CLOCK OPTION

The real-time clock for the Varian 520/i produces a variable-interval interrupt derived from either a crystal oscillator or the power line frequency. (The following paragraphs are written on the assumption that the power line frequency is 60 Hz; if another frequency is used, its value can be substituted for 60 Hz, where appropriate, in the following discussion.)

To produce the variable-interval interrupt, the real-time clock starts with either a 60-Hz line frequency or a 8-MHz signal from a crystal oscillator. The latter is divided to a frequency of 80 kHz, and the 60-Hz or 80-kHz signal is then further divided by the values selected for N or M in the following formula:

$$\text{Rate} = \frac{60}{2^N \times M} \text{ or}$$

$$\frac{80,000}{2^N \times M} \text{ interrupts/second}$$

where:

N = an integer with a value of 0, 1, 2, 3, or 4.

M = an integer, $1 \leq M \leq 255$

The resulting signal drives a counter that resets upon coincidence with the value of M set by program control. The variable-interval

interrupt is also generated each time the coincidence occurs.

The value of M is transmitted to the real-time clock from the CPU via the 8-bit I/O bus. (The value M = 0 disables the variable-interval interrupt.) The value of N is selected by jumper wiring in a special plug-in component called hardwired header.

In addition to the interrupt signals, the real-time clock also produces a set of square waves for timing external devices. The square-wave signals include 4 MHz and 60 Hz, plus one of two series based on divisions of either the 8-kHz crystal-oscillator-derived signal (80 kHz, 40 kHz, 20 kHz, 10 kHz and 5 kHz) or the 60-Hz line-frequency signal (60 Hz, 30 Hz, 15 Hz, 7.5 Hz, and 3.75 Hz).

The 60 Hz signal can be used for keeping time of day, when that function is desired, without affecting the variable-interval interrupt output of the real-time clock.

Functional Description

The block diagram of the real-time clock is presented in Figure 30-1.

Standard Varian 520/i drivers and receivers are used to communicate with the 8-bit I/O bus. Seven additional drivers are included to supply the square-wave outputs. The latter are transmitted through twisted pair and

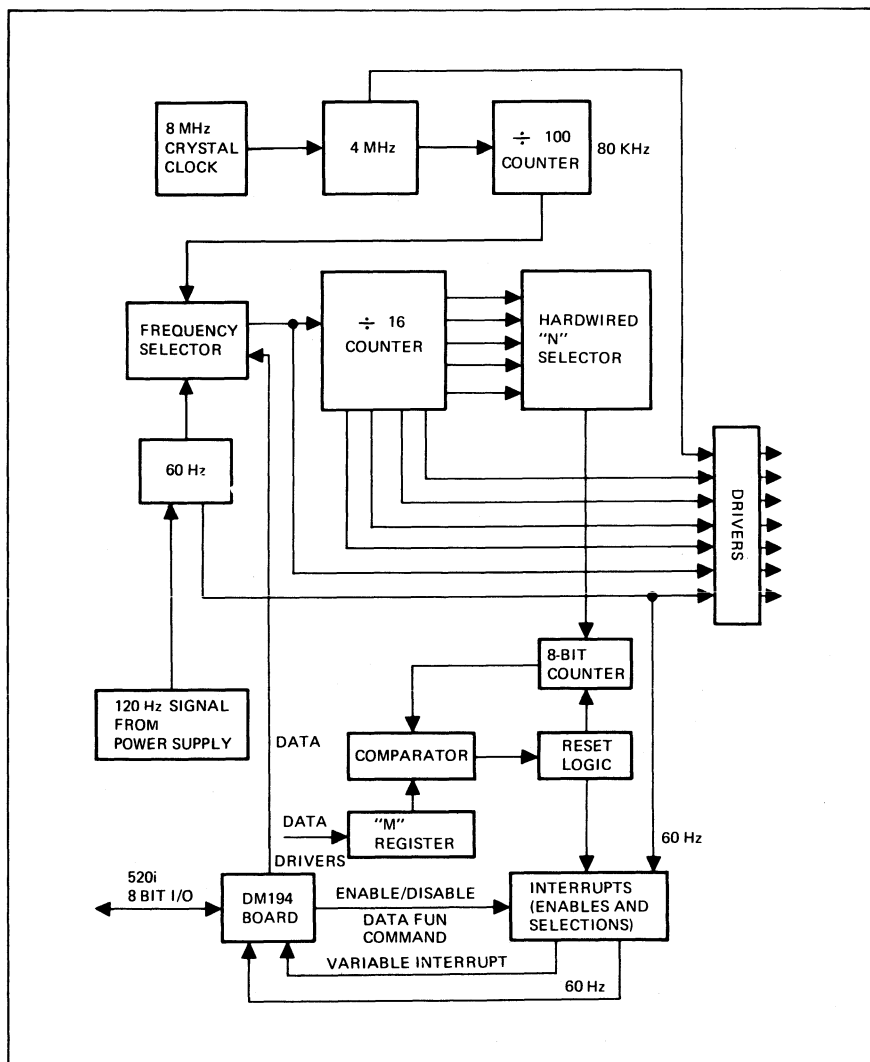


Figure 30-1. Real-Time Clock Block Diagram

terminated with a 130-ohm resistor at +5 V. If terminated in this manner, the outputs are at standard TTL logic levels and each output can drive 25 standard TTL Loads (74 series).

The count-length register, which holds the value of M, is loaded via the 8-bit I/O bus by a BTO command with a device address of \$03.

The 8-MHz crystal oscillator is the basic source for the 8-kHz signal. The source for the 60-Hz signal is the power-line frequency, which is derived from the 120-Hz ripple frequency of the power-supply bridge. The selection of the 8-kHz or 60-Hz signal is under program control.

The divide-by-16 counter is provided with taps at each stage. Each of the five frequencies is fed to a driver which provides a square-wave output. In addition, one of the five frequencies is fed to an 8-bit counter to produce the variable-interval interrupt. The frequency is selected by a hardwired header, which sets the value of "N." (If not specified, the value of "N" is set at 0.)

The 8-bit counter feeds a comparator which initiates reset of the counter when the count corresponds to the value of M set in the count-length register. On coincidence, an interrupt is generated and the 8-bit counter is reset.

Programming the Real-Time Clock

Interrupts are enabled and disabled by function commands from the CPU. The 60 Hz and the variable-interval interrupt may be activated and deactivated separately. The

two interrupt sources may drive any two of the eleven Varian 520/i interrupts. The selection is made using hardwired headers.

Device address for the real time clock is \$ 03. The following instructions are used:

BTO \$	3003	Load M register from the lower 8-bits of the accumulator and reset 8-bit counter.
FUN 03 \$	3303	Enable 60 Hz interrupt.
FUN 23 \$	3323	Disable 60 Hz interrupt.
FUN 43 \$	3343	Reset 60 Hz interrupt.
FUN 63 \$	3363	Enable variable interval interrupt.
FUN 83 \$	3383	Disable variable interval interrupt.
FUN A3 \$	33A3	Reset variable interval interrupt.
FUN C3 \$	33C3	Connect 60 Hz to variable interrupt counter and reset divide-by-16 counter and 8-bit counter.
FUN E3 \$	33E3	Connect 80 kHz to variable interrupt counter and reset divide-by-16 counter and 8-bit counter.

Physical Description

The real-time clock consists of 43 IC chips and an 8-MHz crystal oscillator. Two input cables are required, and one output connection must be provided for the seven reference frequencies.

The power requirements for the option are as follows:

+5 Vdc: 530 mA
+12 Vdc: 64 mA
-12 Vdc: 86 mA

The real-time clock is mounted on a quarter socket board in the Varian 520/i Expansion Chassis. Connections, logic levels and timing conform to the standard 8-bit I/O bus system described in Section 8. The signals from the 8-bit I/O bus are repowered on the DM 220 board in the Expansion Chassis (see Section 26) and then brought into the logic on the real-time clock board.

SECTION 31 — VARIAN 520/DC DATA COMMUNICATION SYSTEM

The Varian 520/DC system brings a new level of versatility and flexibility to data-communication networks.

For the first time, the interchange of data among a variety of terminals and computers can be accomplished under computer control on an individualized, line-by-line basis. Each input/output line, up to a total of 64, is provided with its own processing program. Moreover, the operator can at any time, without shifting a wire or turning a switch, change these individual programs to meet changing operational requirements.

The flexibility of the Varian 520/DC system allows it to be used interchangeably as a data concentrator, linking a number of low-speed lines to one or more high-speed lines, or a communications preprocessor, organizing data for direct entry into a large computer.

In both cases, the varian 520/DC is adaptable to interactive or "conversational" systems involving a frequent exchange of relatively short messages. Typical would be a time-sharing system, or an airlines-reservation network, or any other type of system in which a major computer is time-shared by a number of users.

The Varian 520/DC can accommodate signals from a variety of terminal devices including teletypes, tape readers, card-punch/readers, CRT displays, and keyboards. It can

also interface with all standard common-carrier modems.

The input/output lines can be half or full duplex, synchronous or asynchronous. Data characters can contain 5, 6, 7, or 8 bits, with an odd or even parity check.

Synchronous lines can operate at any rate up to 4800 baud (bits per second). Asynchronous lines may use any one of four clock signals. Each clock rate is pre-set by wiring on a connector to any value up to 1200 baud.

All of these options can be selected and changed on a line-by-line basis, through software via the operator's console.

The Varian 520/DC is modularly designed for economical expansion. Communication-line control modules can be added in increments of four, up to a total of 64. The complete system, including computer, line control modules, and power supplies, is housed in a single five-foot rack.

System Description

The Varian 520/DC system consists of three major elements: one or more Line Control Modules (one for each four input/output lines), a Varian 520/i-60 Communication Controller, modularly expandable to accommodate 64 lines, and a Varian 520/i digital computer (see Figure 31-1).

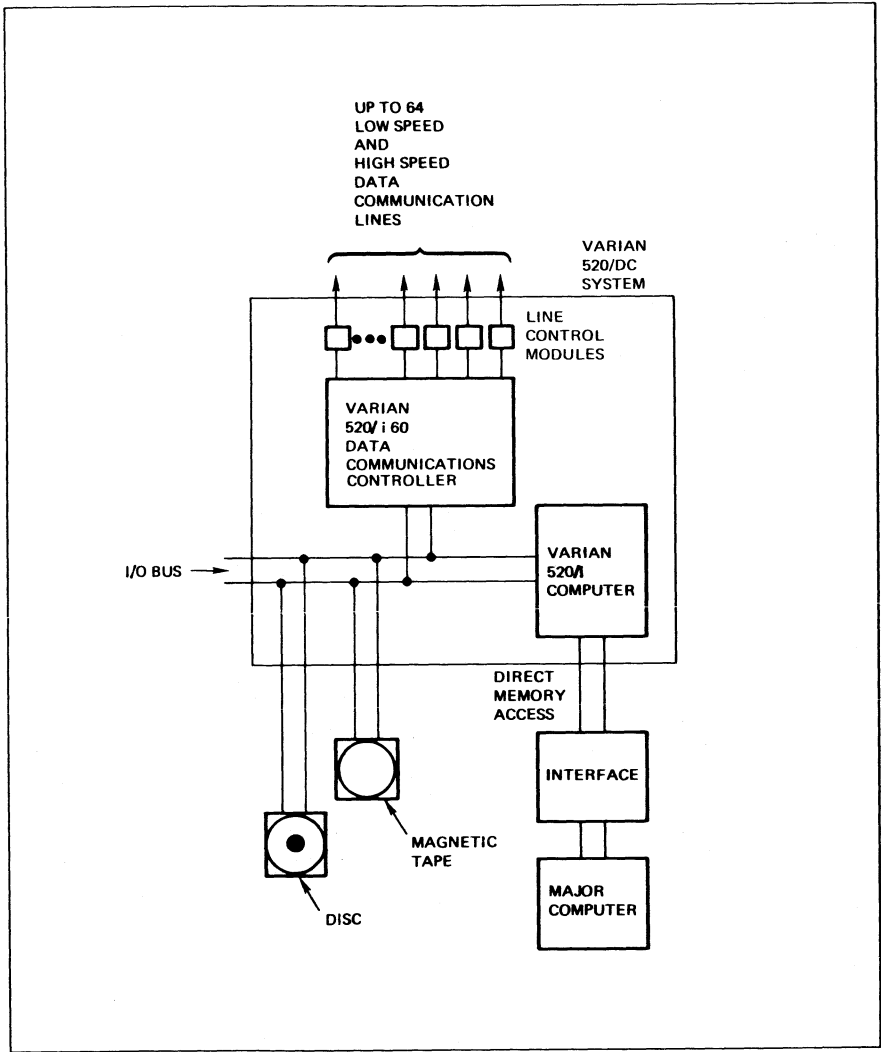


Figure 31-1. Varian 520/DC Data Communication System

The Line Control Modules accept and transmit data on a serial, bit-by-bit basis, serving primarily as electrical interfaces for the terminal devices or common-carrier modems linked to the system.

The Varian 520/i-60 controller samples the condition of each line every 58 microseconds. Using these samples, the controller establishes synchronization and assembles bits into characters of received data. When transmitting, the controller disassembles the characters into bits. The individual characters can contain 5, 6, 7, or 8 bits, plus an odd or even parity bit, in accordance with the control program for that particular line. The control program can itself also be changed at each sample time.

Assembled characters are transferred between the Varian 520/i-60 controller and the Varian 520/i computer via the computer's I/O bus. These transfers are made under interrupt control whenever the controller senses that a complete character has been assembled.

The transfers consist of two 8-bit bytes. One byte carries the data, the other byte contains the communication-line address and two control bits indicating the nature of the data being transferred.

Normally the data is information being received or transmitted by the system. But the data can also be the control information affecting the processing of data on a particular line. Thus, at any time during the operation of the system, it is possible to change communication-line parameters such as the data rate, character length, and type of parity test.

The Varian 520/i computer assembles the characters from each communication line into message blocks. The length of each block is variable, under software control.

The communication controller is only one of the elements that can be attached on the Varian 520/i I/O bus; the system can also include a variety of peripheral devices for bulk storage or other processing functions. Typical would be the addition of magnetic tape transports or a disk.

When the Varian 520/DC system is used as a communications preprocessor for a major computer, the Varian 520/i computer is equipped with a Direct Memory Access (DMA) option and computer-to-computer interface so that data transfers can be made directly from core memory at rates up to 666,000 bytes per second.

Varian 520/i-60 Controller

The Varian 520/i-60 controller contains a number of key elements that contribute to the exceptional flexibility of the Varian 520/DC system.

The most important of these is the use of an integrated-circuit "memory" to store the control information and data for each communication line. Each communication-line memory is capable of storing an 86-bit word. A complete 64-line system therefore contains 64 such 86-bit words.

Figure 31-2 details the information contained in a single Line Control Word. There are four data fields, one for assembling an incoming character, one for disassembling an outgoing character, and two buffers for

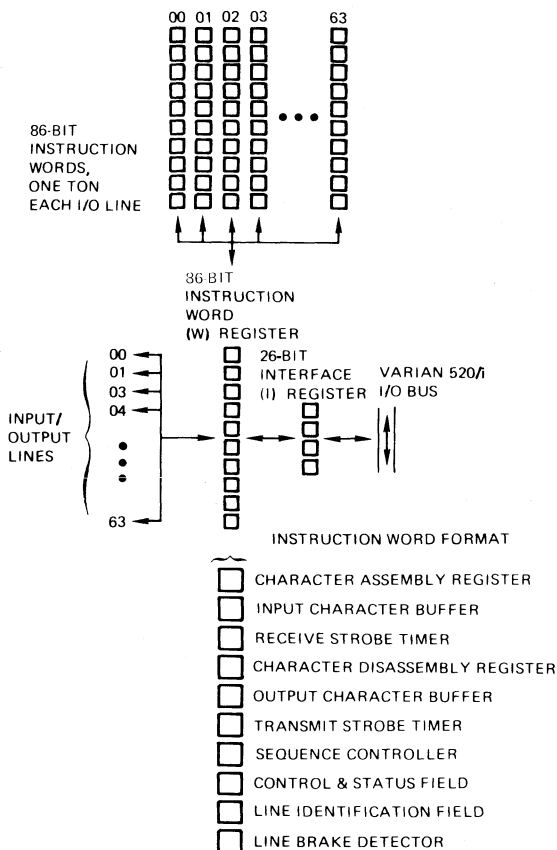


Figure 31-2. Varian 520/i-60 Data Communication Controller

holding a complete character that has been assembled or is ready to be disassembled.

Other Line Control Word fields are interpreted by the controller to determine and control the characteristics of that particular communication line. Parameters include character length (5, 6, 7, or 8 bits), baud rate, synchronous or asynchronous transmission, and type of parity test.

The controller "updates" the individual Line Control Words by transferring them, one at a time, into the 86-bit W Register. The transfers in and out of the W Register take place at a 1.1 MHz rate, which means that once every 58 microseconds, each word occupies the W Register for 900 nanoseconds.

During this 900 nanoseconds, the corresponding Line Control Module is also connected to the W Register and a single bit is transferred (if appropriate) in or out of the assembly or disassembly register. At the same time complete characters are transferred (if appropriate) to or from the Interface Register for transmission on the I/O bus. At the end of the 900 nanoseconds, the Line Control Word is returned to its own memory position, altered by the addition or subtraction of individual bits or characters.

The Interface Register is also the vehicle for transferring program changes to each of the Line Control Words. During each Word's 900 nanosecond occupancy of the W Register, the Interface Register can write new control information affecting such parameters as word length and baud rate.

The Interface Register communicates with the Varian 520/i computer through the

standard interrupt and sensing system. When the Interface Register has received an assembled character from the W Register, it generates an interrupt signal. The computer interrupt routine then inputs the character and line number from the interface register. The same interrupt is used to request a character from the computer for an output transmission. A second type of interrupt indicates that an error has occurred in the data-transmission process.

When the computer program wants to initiate a data transfer to the controller, it senses the state of the Interface Register. If it is not "busy," an address-and-data message is transmitted via the I/O bus to the Interface Register, where it is held until the addressed Line Control Word cycles through the W Register.

Software

Three software packages are supplied with the Varian 520/DC system.

- The on line operating program required by the "foreground" environment processing the I/O data transfers.
- The operator's setup program for entering communication-line parameters via the teletype keyboard.
- A diagnostic program for testing the operation of the complete system.

The program for the data processing "background" environment is written by the user, employing standard Varian 520/i programming techniques.

The background environment is interruptible; the foreground environment is not. In normal operation, the computer remains in the background environment until an I/O interrupt is received from the controller. The program then switches to the foreground environment and carries through, without interruption, the I/O data transfer for one character. On completion of the transfer, program control is returned to the background environment.

Control Stacks

The foreground program includes 27-byte control stacks, one for each data-communication line. The control stack has the following information about its associated communication line:

- Last input line address word
- Last input character
- Line identification, including
 - Baud rate (asynchronous only)
 - Synchronous or asynchronous
 - Bits/character
 - Parity, even or odd
 - Priority, high or low
 - Stop bits, one or two
- Longitudinal parity character
- Input message buffer start address
- Input character counter
- Input message buffer length
- Address of buffer-full subroutine
- Address of control character subroutine
- Synchronous code flag

- Control character recognition mask
- Address of input error subroutine
- Output message buffer start address
- Output character counter
- Output message buffer length
- Address of buffer empty subroutine
- Address of line error subroutine
- Address of ring subroutine
- I/O mode (mirror image, etc.)

System Specifications

Line Configuration

Capacity Range

Four channels (minimum), up to 64 channels (maximum)

Expansion

Plug-in expandable in 4-channel increments up to the maximum line capacity.

Line Dynamics

Transmission Type

Half- or full-duplex selected for any line channel under software control.

Transmission Mode

Synchronous — Speeds from 45 up to 4800 baud. Only 8-bit character code format is acceptable.

Asynchronous — Speeds from 45 up to 1200 baud. Four user selectable speed

values for any channel under program control (values determined by connector wiring). One start and one, one-and-one-half, or two stop bits. Variable size character codes using 5, 6, 7, or 8 bits per character.

Line Program Features:

Parity

Checked on incoming data lines and generated for outgoing data lines (odd or even) under software control.

Priority

Under software control, any line (or lines) may be designated as high priority.

Line Interface:

Design Structure

Line control module is available that is plug-in compatible with Western Electric Modems 103, 201, 202, or commercial equivalents; other modules are compatible with asynchronous two-wire system; asynchronous relay systems; and synchronous ground-isolated systems.

Interconnection

Modem cables of up to 50 feet can be used.

Call-Up Capability

Western Electric 801A Automatic Call Unit or commercial equivalent.

APPENDIX INDEX

	Page
Two's Complement Arithmetic	A-2
Power of 2, Base 10, Table A-1	A-3
Introduction to Hexadecimal Arithmetic	A-4
Binary/Hexadecimal/Decimal Conversions, Table A-2	A-5
Hexadecimal/Decimal Large Integer Conversions, Table A-3	A-6
Hexadecimal/Decimal Integer Conversions, Table A-4	A-7
Hexadecimal/Decimal Fraction Conversions, Table A-5	A-13
Hexadecimal Arithmetic, Table A-6	A-29
Mathematical Constants, Table A-7	A-30
Powers of 16, Base 10, Table A-8	A-31
Powers of 10, Base 16, Table A-9	A-32
Standard Subroutine Running Times	A-33
Binary Object Tape Format	A-34
Standard Character Codes	A-36
Assembly System Error Characters	A-38
Peripheral Addresses	A-39
Index of Symbolic Terms	A-40
Memory-Reference Instructions	A-42
Non-Memory One-Byte Instructions	A-44
Register-Operate Instructions	A-44
Control Instructions	A-46
Skip Instructions	A-48
Shift Instructions	A-51
Environmental Control Instructions	A-53
Non-Memory Reference Two-Byte Instructions	A-54
Register-Operate Instructions	A-54
Input/Output Instructions	A-57
Assigned Input/Output Instructions	A-59
Teletype Controller Reserved Instructions	A-62
Non-Memory-Reference, Three-Byte Instructions	A-65
Pseudo Instructions	A-66
Alphabetical Index of Instructions	A-69
Numerical Index of Instructions	A-76

Two's Complement Arithmetic

Binary numbers in the Varian 520/i may have a precision of eight bits, 16 bits, 24 bits or 32 bits. In each case, the most-significant bit is the sign bit. A sign bit of ZERO denotes a positive number; a sign bit of ONE denotes a negative number. The negative of a positive number is represented in 2's complement form.

The 2's complement of a number may be found in either of two ways:

Take the 1's complement of the number (complement each bit) and add 1 to the result.
Example:

number	00001001	(+9)
1's complement	11110110	
	+1	
2's complement	11110111	(-9)

For an n-bit number (including the sign bit), subtract the number from $2^{(n+1)}$.
Example:

$2^{(n+1)}$	100000000
-(+9)	-00001001
-9	11110111

Table A-1. Powers of Two, Base 10

2^{η}	η	2^{η}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	12	0.000 12 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125

Hexadecimal Arithmetic

It is often convenient to express binary numbers by their hexadecimal equivalent.

Such conversions are easily performed by grouping the binary bits in fours, starting with the least-significant bit. In this way, numbers with an eight-bit precision may be expressed by only two hexadecimal digits, 0 0 to F F. For the maximum Varian 520/i precision of 32 bits, only eight hexadecimal digits are required.

The range of numbers in the 520/i is from $-2^{(31)}$ to $2^{(31)}-1$. The numbers zero, minus 1, and plus and minus full scale are shown below.

Decimal	Hexadecimal
2,147,483,647 (+full scale)	7 F F F F F F F
0	0 0 0 0 0 0 0 0
-1	F F F F F F F F
-2,147,483,648 (-full scale)	8 0 0 0 0 0 0 0

In performing addition or subtraction, it is possible for the result to exceed the full-scale range of the machine. For example:

Decimal	Hexadecimal
83	53
+79	+4F
163	A2(-94)

The negative hexadecimal result is in error. The same type of error occurs when the sum of two negative numbers exceeds the negative full-scale range:

Decimal	Hexidecimal
-83	AD
(+)-79	+B1
-162	(1) 5E (+94)

The carry bit out of the most significant data position is lost; however, to inform the programmer that the result of the operation is outside the range of the set precision, an

overflow indicator is provided (one for each environment). The overflow indicator is set if the sign bit changes when two numbers of the same sign are added.

The following pages provide the user with tables for converting digital values and commonly used constants to an equivalent hexadecimal value. Table A-2 presents direct conversions between hexadecimal integers 000 to FFF and decimal integers 0000 to 4095. For conversion of larger integers, Table A-3 presents values that may be added to those of Table A-4. Table A-5 presents direct conversions between hexadecimal and decimal fractions.

Table A-2. Binary/Hexadecimal/Decimal Conversions

Binary	Hexadecimal	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Table A-3. Hexadecimal/Decimal Large Integer Conversions

Hexadecimal	Decimal	Hexadecimal	Decimal
01 000	4 096	20 000	131 072
02 000	8 192	30 000	196 608
03 000	12 288	40 000	262 144
04 000	16 384	50 000	327 680
05 000	20 480	60 000	393 216
06 000	24 576	70 000	458 752
07 000	28 672	80 000	524 288
08 000	32 768	90 000	589 824
09 000	36 864	A0 000	655 360
0A 000	40 960	B0 000	720 896
0B 000	45 056	C0 000	786 432
0C 000	49 152	D0 000	851 968
0D 000	53 248	E0 000	917 504
0E 000	57 344	F0 000	983 040
0F 000	61 440	100 000	1 048 576
10 000	65 536	200 000	2 097 152
11 000	69 632	300 000	3 145 728
12 000	73 728	400 000	4 194 304
13 000	77 824	500 000	5 242 880
14 000	81 920	600 000	6 291 456
15 000	86 016	700 000	7 340 032
16 000	90 112	800 000	8 388 608
17 000	94 208	900 000	9 437 184
18 000	98 304	A00 000	10 485 760
19 000	102 400	B00 000	11 534 336
1A 000	106 496	C00 000	12 582 912
1B 000	110 592	D00 000	13 631 488
1C 000	114 688	E00 000	14 680 064
1D 000	118 784	F00 000	15 728 640
1E 000	122 880	1 000 000	16 777 216
1F 000	126 976	2 000 000	33 554 432

Table A-4. Hexadecimal/Decimal Integer Conversions

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
010	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
020	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
030	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
040	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
050	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
060	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
070	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
080	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
090	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A0	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B0	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C0	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D0	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E0	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F0	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255
100	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
110	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
120	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
130	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
140	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
150	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
160	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
170	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
180	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
190	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A0	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B0	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C0	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D0	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E0	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F0	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
200	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
210	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
220	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
230	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
240	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
250	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
260	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
270	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
280	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
290	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A0	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B0	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C0	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D0	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E0	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F0	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767

Table A-4. Hexadecimal/Decimal Integer Conversions (Continued)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
300	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
310	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
320	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
330	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
340	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
350	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
360	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
370	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
380	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
390	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A0	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B0	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C0	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D0	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E0	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F0	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023
400	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
410	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
420	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
430	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
440	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
450	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
460	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
470	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
480	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
490	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A0	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B0	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C0	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D0	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E0	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F0	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
500	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
510	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
520	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
530	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
540	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
550	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
560	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
570	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
580	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
590	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A0	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B0	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C0	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D0	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E0	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F0	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535

Table A-4. Hexadecimal/Decimal Integer Conversions (Continued)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
600	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
610	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
620	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
630	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
640	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
650	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
660	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
670	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
680	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
690	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A0	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B0	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C0	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D0	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E0	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F0	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791
700	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
710	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
720	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
730	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
740	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
750	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
760	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
770	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
780	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
790	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A0	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B0	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C0	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D0	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E0	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F0	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
800	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
810	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
820	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
830	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
840	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
850	2118	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
860	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
870	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
880	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
890	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A0	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B0	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C0	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D0	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E0	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F0	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303

Table A-4. Hexadecimal/Decimal Integer Conversions (Continued)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
900	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
910	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
920	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
930	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
940	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
950	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
960	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
970	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
980	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
990	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A0	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B0	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C0	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D0	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E0	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F0	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559
A00	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A10	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A20	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A30	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A40	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A50	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A60	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A70	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A80	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A90	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA0	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB0	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC0	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD0	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE0	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF0	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B00	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B10	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B20	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B30	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B40	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B50	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B60	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B70	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B80	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B90	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA0	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB0	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC0	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD0	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE0	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF0	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071

Table A-4. Hexadecimal/Decimal Integer Conversions (Continued)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C00	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C10	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C20	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C30	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C40	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C50	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C60	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C70	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C80	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C90	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA0	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB0	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC0	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD0	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE0	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF0	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327
D00	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D10	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D20	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D30	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D40	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D50	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D60	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D70	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D80	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D90	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA0	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB0	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC0	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD0	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE0	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF0	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
E00	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E10	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E20	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E30	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E40	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E50	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E60	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E70	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E80	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E90	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA0	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB0	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC0	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED0	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE0	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF0	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839

Table A-4. Hexadecimal/Decimal Integer Conversions (Continued)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
F00	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F10	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F20	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	2883	3884	3885	3886	3887
F30	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F40	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F50	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F60	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F70	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F80	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F90	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA0	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB0	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC0	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD0	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE0	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF0	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

Table A-5. Hexadecimal/Decimal Fraction Conversions

Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 00	.00000 00000	.20 00 00 00	.12500 00000
.01 00 00 00	.00390 62500	.21 00 00 00	.12890 62500
.02 00 00 00	.00781 25000	.22 00 00 00	.13281 25000
.03 00 00 00	.01171 87500	.23 00 00 00	.13671 87500
.04 00 00 00	.01562 50000	.24 00 00 00	.14062 50000
.05 00 00 00	.01953 12500	.25 00 00 00	.14453 12500
.06 00 00 00	.02343 75000	.26 00 00 00	.14843 75000
.07 00 00 00	.02734 37500	.27 00 00 00	.15234 37500
.08 00 00 00	.03125 00000	.28 00 00 00	.15625 00000
.09 00 00 00	.03515 62500	.29 00 00 00	.16015 62500
.0A 00 00 00	.03906 25000	.2A 00 00 00	.16406 25000
.0B 00 00 00	.04296 87500	.2B 00 00 00	.16796 87500
.0C 00 00 00	.04687 50000	.2C 00 00 00	.17187 50000
.0D 00 00 00	.05078 12500	.2D 00 00 00	.17578 12500
.0E 00 00 00	.05468 75000	.2E 00 00 00	.17968 75000
.0F 00 00 00	.05859 37500	.2F 00 00 00	.18359 37500
.10 00 00 00	.06250 00000	.30 00 00 00	.18750 00000
.11 00 00 00	.06640 62500	.31 00 00 00	.19140 62500
.12 00 00 00	.07031 25000	.32 00 00 00	.19531 25000
.13 00 00 00	.07421 87500	.33 00 00 00	.19921 87500
.14 00 00 00	.07812 50000	.34 00 00 00	.20312 50000
.15 00 00 00	.08203 12500	.35 00 00 00	.20703 12500
.16 00 00 00	.08593 75000	.36 00 00 00	.21093 75000
.17 00 00 00	.08984 37500	.37 00 00 00	.21484 37500
.18 00 00 00	.09375 00000	.38 00 00 00	.21875 00000
.19 00 00 00	.09765 62500	.39 00 00 00	.22265 62500
.1A 00 00 00	.10156 25000	.3A 00 00 00	.22656 25000
.1B 00 00 00	.10546 87500	.3B 00 00 00	.23046 87500
.1C 00 00 00	.10937 50000	.3C 00 00 00	.23437 50000
.1D 00 00 00	.11328 12500	.3D 00 00 00	.23828 12500
.1E 00 00 00	.11718 75000	.3E 00 00 00	.24218 75000
.1F 00 00 00	.12109 37500	.3F 00 00 00	.24609 37500

Table A-5. Hexadecimal/Decimal Fraction Conversions (Continued)

Hexadecimal	Decimal	Hexadecimal	Decimal
.40 00 00 00	.25000 00000	.60 00 00 00	.37500 00000
.41 00 00 00	.25390 62500	.61 00 00 00	.37890 62500
.42 00 00 00	.25781 25000	.62 00 00 00	.38281 25000
.43 00 00 00	.26171 87500	.63 00 00 00	.38671 87500
.44 00 00 00	.26562 50000	.64 00 00 00	.39062 50000
.45 00 00 00	.26953 12500	.65 00 00 00	.39453 12500
.46 00 00 00	.27343 75000	.66 00 00 00	.39843 75000
.47 00 00 00	.27734 37500	.67 00 00 00	.40234 37500
.48 00 00 00	.28125 00000	.68 00 00 00	.40625 00000
.49 00 00 00	.28515 62500	.69 00 00 00	.41015 62500
.4A 00 00 00	.28906 25000	.6A 00 00 00	.41406 25000
.4B 00 00 00	.29296 87500	.6B 00 00 00	.41796 87500
.4C 00 00 00	.29687 50000	.6C 00 00 00	.42187 50000
.4D 00 00 00	.30078 12500	.6D 00 00 00	.42578 12500
.4E 00 00 00	.30468 75000	.6E 00 00 00	.42968 75000
.4F 00 00 00	.30859 37500	.6F 00 00 00	.43359 37500
.50 00 00 00	.31250 00000	.70 00 00 00	.43750 00000
.51 00 00 00	.31640 62500	.71 00 00 00	.44140 62500
.52 00 00 00	.32031 25000	.72 00 00 00	.44531 25000
.53 00 00 00	.32421 87500	.73 00 00 00	.44921 87500
.54 00 00 00	.32812 50000	.74 00 00 00	.45312 50000
.55 00 00 00	.33203 12500	.75 00 00 00	.45703 12500
.56 00 00 00	.33593 75000	.76 00 00 00	.46093 75000
.57 00 00 00	.33984 37500	.77 00 00 00	.46484 37500
.58 00 00 00	.34375 00000	.78 00 00 00	.46875 00000
.59 00 00 00	.34765 62500	.79 00 00 00	.47265 62500
.5A 00 00 00	.35156 25000	.7A 00 00 00	.47656 25000
.5B 00 00 00	.35546 87500	.7B 00 00 00	.48046 87500
.5C 00 00 00	.35937 50000	.7C 00 00 00	.48437 50000
.5D 00 00 00	.36328 12500	.7D 00 00 00	.48828 12500
.5E 00 00 00	.36718 75000	.7E 00 00 00	.49218 75000
.5F 00 00 00	.37109 37500	.7F 00 00 00	.49609 37500

Table A-5. Hexadecimal/Decimal Fraction Conversions (Continued)

Hexadecimal	Decimal	Hexadecimal	Decimal
.80 00 00 00	.50000 00000	.A0 00 00 00	.62500 00000
.81 00 00 00	.50390 62500	.A1 00 00 00	.62890 62500
.82 00 00 00	.50781 25000	.A2 00 00 00	.63281 25000
.83 00 00 00	.51171 87500	.A3 00 00 00	.63671 87500
.84 00 00 00	.51562 50000	.A4 00 00 00	.64062 50000
.85 00 00 00	.51953 12500	.A5 00 00 00	.64453 12500
.86 00 00 00	.52343 75000	.A6 00 00 00	.64843 75000
.87 00 00 00	.52734 37500	.A7 00 00 00	.65234 37500
.88 00 00 00	.53125 00000	.A8 00 00 00	.65625 00000
.89 00 00 00	.53515 62500	.A9 00 00 00	.66015 62500
.8A 00 00 00	.53906 25000	.AA 00 00 00	.66406 25000
.8B 00 00 00	.54296 87500	.AB 00 00 00	.66796 87500
.8C 00 00 00	.54687 50000	.AC 00 00 00	.67187 50000
.8D 00 00 00	.55078 12500	.AD 00 00 00	.67578 12500
.8E 00 00 00	.55468 75000	.AE 00 00 00	.67968 75000
.8F 00 00 00	.55859 37500	.AF 00 00 00	.68359 37500
.90 00 00 00	.56250 00000	.B0 00 00 00	.68750 00000
.91 00 00 00	.56640 62500	.B1 00 00 00	.69140 62500
.92 00 00 00	.57031 25000	.B2 00 00 00	.69531 25000
.93 00 00 00	.57421 87500	.B3 00 00 00	.69921 87500
.94 00 00 00	.57812 50000	.B4 00 00 00	.70312 50000
.95 00 00 00	.58203 12500	.B5 00 00 00	.70703 12500
.96 00 00 00	.58593 75000	.B6 00 00 00	.71093 75000
.97 00 00 00	.58984 37500	.B7 00 00 00	.71484 37500
.98 00 00 00	.59375 00000	.B8 00 00 00	.71875 00000
.99 00 00 00	.59765 62500	.B9 00 00 00	.72265 62500
.9A 00 00 00	.60156 25000	.BA 00 00 00	.72656 25000
.9B 00 00 00	.60546 87500	.BB 00 00 00	.73046 87500
.9C 00 00 00	.60937 50000	.BC 00 00 00	.73437 50000
.9D 00 00 00	.61328 12500	.BD 00 00 00	.73828 12500
.9E 00 00 00	.61718 75000	.BE 00 00 00	.74218 75000
.9F 00 00 00	.62109 37500	.BF 00 00 00	.74609 37500

Table A-5. Hexadecimal/Decimal Fraction Conversions (Continued)

Hexadecimal	Decimal	Hexadecimal	Decimal
.C0 00 00 00	.75000 00000	.E0 00 00 00	.87500 00000
.C1 00 00 00	.75390 62500	.E1 00 00 00	.87890 62500
.C2 00 00 00	.75781 25000	.E2 00 00 00	.88281 25000
.C3 00 00 00	.76171 87500	.E3 00 00 00	.88671 87500
.C4 00 00 00	.76562 50000	.E4 00 00 00	.89062 50000
.C5 00 00 00	.76953 12500	.E5 00 00 00	.89453 12500
.C6 00 00 00	.77343 75000	.E6 00 00 00	.89843 75000
.C7 00 00 00	.77734 37500	.E7 00 00 00	.90234 37500
.C8 00 00 00	.78125 00000	.E8 00 00 00	.90625 00000
.C9 00 00 00	.78515 62500	.E9 00 00 00	.91015 62500
.CA 00 00 00	.78906 25000	.EA 00 00 00	.91406 25000
.CB 00 00 00	.79296 87500	.EB 00 00 00	.91796 87500
.CC 00 00 00	.79687 50000	.EC 00 00 00	.92187 50000
.CD 00 00 00	.80078 12500	.ED 00 00 00	.92578 12500
.CE 00 00 00	.80468 75000	.EE 00 00 00	.92968 75000
.CF 00 00 00	.80859 37500	.EF 00 00 00	.93359 37500
.D0 00 00 00	.81250 00000	.F0 00 00 00	.93750 00000
.D1 00 00 00	.81640 62500	.F1 00 00 00	.94140 62500
.D2 00 00 00	.82031 25000	.F2 00 00 00	.94531 25000
.D3 00 00 00	.82421 87500	.F3 00 00 00	.94921 87500
.D4 00 00 00	.82812 50000	.F4 00 00 00	.95312 50000
.D5 00 00 00	.83203 12500	.F5 00 00 00	.95703 12500
.D6 00 00 00	.83593 75000	.F6 00 00 00	.96093 75000
.D7 00 00 00	.83984 37500	.F7 00 00 00	.96484 37500
.D8 00 00 00	.84375 00000	.F8 00 00 00	.96875 00000
.D9 00 00 00	.84765 62500	.F9 00 00 00	.97265 62500
.DA 00 00 00	.85156 25000	.FA 00 00 00	.97656 25000
.DB 00 00 00	.85546 87500	.FB 00 00 00	.98046 87500
.DC 00 00 00	.85937 50000	.FC 00 00 00	.98437 50000
.DD 00 00 00	.86328 12500	.FD 00 00 00	.98828 12500
.DE 00 00 00	.86718 75000	.FE 00 00 00	.99218 75000
.DF 00 00 00	.87109 37500	.FF 00 00 00	.99609 37500

Table A-5. Hexadecimal/Decimal Fraction Conversions (Continued)

Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 00	.00000 00000	.00 20 00 00	.00048 82812
.00 01 00 00	.00001 52587	.00 21 00 00	.00050 35400
.00 02 00 00	.00003 05175	.00 22 00 00	.00051 87988
.00 03 00 00	.00004 57763	.00 23 00 00	.00053 40576
.00 04 00 00	.00006 10351	.00 24 00 00	.00054 93164
.00 05 00 00	.00007 62939	.00 25 00 00	.00056 45751
.00 06 00 00	.00009 15527	.00 26 00 00	.00057 98339
.00 07 00 00	.00010 68115	.00 27 00 00	.00059 50927
.00 08 00 00	.00012 20703	.00 28 00 00	.00061 03515
.00 09 00 00	.00013 73291	.00 29 00 00	.00062 56103
.00 0A 00 00	.00015 25878	.00 2A 00 00	.00064 08691
.00 0B 00 00	.00016 78466	.00 2B 00 00	.00065 61279
.00 0C 00 00	.00018 31054	.00 2C 00 00	.00067 13867
.00 0D 00 00	.00019 83642	.00 2D 00 00	.00068 66455
.00 0E 00 00	.00021 36230	.00 2E 00 00	.00070 19042
.00 0F 00 00	.00022 88818	.00 2F 00 00	.00071 71630
.00 10 00 00	.00024 41406	.00 30 00 00	.00073 24218
.00 11 00 00	.00025 93994	.00 31 00 00	.00074 76806
.00 12 00 00	.00027 46582	.00 32 00 00	.00076 29394
.00 13 00 00	.00028 99169	.00 33 00 00	.00077 81982
.00 14 00 00	.00030 51757	.00 34 00 00	.00079 34570
.00 15 00 00	.00032 04345	.00 35 00 00	.00080 87158
.00 16 00 00	.00033 56933	.00 36 00 00	.00082 39746
.00 17 00 00	.00035 09521	.00 37 00 00	.00083 92333
.00 18 00 00	.00036 62109	.00 38 00 00	.00085 44921
.00 19 00 00	.00038 14697	.00 39 00 00	.00086 97509
.00 1A 00 00	.00039 67285	.00 3A 00 00	.00088 50097
.00 1B 00 00	.00041 19873	.00 3B 00 00	.00090 02685
.00 1C 00 00	.00042 72460	.00 3C 00 00	.00091 55273
.00 1D 00 00	.00044 25048	.00 3D 00 00	.00093 07861
.00 1E 00 00	.00045 77636	.00 3E 00 00	.00094 60449
.00 1F 00 00	.00047 30224	.00 3F 00 00	.00096 13037

Table A-5. Hexadecimal/Decimal Fraction Conversions (Continued)

Hexadecimal	Decimal	Hexadecimal	Decimal
.00 40 00 00	.00097 65625	.00 60 00 00	.00146 48437
.00 41 00 00	.00099 18212	.00 61 00 00	.00148 01025
.00 42 00 00	.00100 70800	.00 62 00 00	.00149 53613
.00 43 00 00	.00102 23388	.00 63 00 00	.00151 06201
.00 44 00 00	.00103 75976	.00 64 00 00	.00152 58789
.00 45 00 00	.00105 28564	.00 65 00 00	.00154 11376
.00 46 00 00	.00106 81152	.00 66 00 00	.00155 63964
.00 47 00 00	.00108 33740	.00 67 00 00	.00157 16552
.00 48 00 00	.00109 86328	.00 68 00 00	.00158 69140
.00 49 00 00	.00111 38916	.00 69 00 00	.00160 21728
.00 4A 00 00	.00112 91503	.00 6A 00 00	.00161 74316
.00 4B 00 00	.00114 44091	.00 6B 00 00	.00163 26904
.00 4C 00 00	.00115 96679	.00 6C 00 00	.00164 79492
.00 4D 00 00	.00117 49267	.00 6D 00 00	.00166 32080
.00 4E 00 00	.00119 01855	.00 6E 00 00	.00167 84667
.00 4F 00 00	.00120 54443	.00 6F 00 00	.00169 37255
.00 50 00 00	.00122 07031	.00 70 00 00	.00170 89843
.00 51 00 00	.00123 59619	.00 71 00 00	.00172 42431
.00 52 00 00	.00125 12207	.00 72 00 00	.00173 95019
.00 53 00 00	.00126 64794	.00 73 00 00	.00175 47607
.00 54 00 00	.00128 17382	.00 74 00 00	.00177 00195
.00 55 00 00	.00129 69970	.00 75 00 00	.00178 52783
.00 56 00 00	.00131 22558	.00 76 00 00	.00180 05371
.00 57 00 00	.00132 75146	.00 77 00 00	.00181 57958
.00 58 00 00	.00134 27734	.00 78 00 00	.00183 10546
.00 59 00 00	.00135 80322	.00 79 00 00	.00184 63134
.00 5A 00 00	.00137 32910	.00 7A 00 00	.00186 15722
.00 5B 00 00	.00138 85498	.00 7B 00 00	.00187 68310
.00 5C 00 00	.00140 38085	.00 7C 00 00	.00189 20898
.00 5D 00 00	.00141 90673	.00 7D 00 00	.00190 73486
.99 5E 00 00	.00143 43261	.00 7E 00 00	.00192 26074
.00 5F 00 00	.00144 95849	.00 7F 00 00	.00193 78662

Table A-5. Hexadecimal/Decimal Fraction Conversions (Continued)

Hexadecimal	Decimal	Hexadecimal	Decimal
.00 80 00 00	.00195 31250	.00 A0 00 00	.00244 14062
.00 81 00 00	.00196 83837	.00 A1 00 00	.00245 66650
.00 82 00 00	.00198 36425	.00 A2 00 00	.00247 19238
.00 83 00 00	.00199 89013	.00 A3 00 00	.00248 71826
.00 84 00 00	.00201 41601	.00 A4 00 00	.00250 24414
.00 85 00 00	.00202 94189	.00 A5 00 00	.00251 77001
.00 86 00 00	.00204 46777	.00 A6 00 00	.00253 29589
.00 87 00 00	.00205 99365	.00 A7 00 00	.00254 82177
.00 88 00 00	.00207 51953	.00 A8 00 00	.00256 34765
.00 89 00 00	.00209 04541	.00 A9 00 00	.00257 87353
.00 8A 00 00	.00210 57128	.00 AA 00 00	.00259 39941
.00 8B 00 00	.00212 09716	.00 AB 00 00	.00260 92529
.00 8C 00 00	.00213 62304	.00 AC 00 00	.00262 45117
.00 8D 00 00	.00215 14892	.00 AD 00 00	.00263 97705
.00 8E 00 00	.00216 67480	.00 AE 00 00	.00265 50292
.00 8F 00 00	.00218 20068	.00 AF 00 00	.00267 02880
.00 90 00 00	.00219 72656	.00 B0 00 00	.00268 55468
.00 91 00 00	.00221 25244	.00 B1 00 00	.00270 08056
.00 92 00 00	.00222 77832	.00 B2 00 00	.00271 60644
.00 93 00 00	.00224 30419	.00 B3 00 00	.00273 13232
.00 94 00 00	.00225 83007	.00 B4 00 00	.00274 65820
.00 95 00 00	.00227 35595	.00 B5 00 00	.00276 18408
.00 96 00 00	.00228 88183	.00 B6 00 00	.00277 70996
.00 97 00 00	.00230 40771	.00 B7 00 00	.00279 23583
.00 98 00 00	.00231 93359	.00 B8 00 00	.00280 76171
.00 99 00 00	.00233 45947	.00 B9 00 00	.00282 28759
.00 9A 00 00	.00234 98535	.00 BA 00 00	.00283 81347
.00 9B 00 00	.00236 51123	.00 BB 00 00	.00285 33935
.00 9C 00 00	.00238 03710	.00 BC 00 00	.00286 86523
.00 9D 00 00	.00239 56298	.00 BD 00 00	.00288 39111
.00 9E 00 00	.00241 08886	.00 BE 00 00	.00289 91699
.00 9F 00 00	.00242 61474	.00 BF 00 00	.00291 44287

Table A-5. Hexadecimal/Decimal Fraction Conversions (Continued)

Hexadecimal	Decimal	Hexadecimal	Decimal
.00 C0 00 00	.00292 96875	.00 E0 00 00	.00341 79687
.00 C1 00 00	.00294 49462	.00 E1 00 00	.00343 32275
.00 C2 00 00	.00296 02050	.00 E2 00 00	.00344 84863
.00 C3 00 00	.00297 54638	.00 E3 00 00	.00346 37451
.00 C4 00 00	.00299 07226	.00 E4 00 00	.00347 90039
.00 C5 00 00	.00300 59814	.00 E5 00 00	.00349 42626
.00 C6 00 00	.00302 12402	.00 E6 00 00	.00350 95214
.00 C7 00 00	.00303 64990	.00 E7 00 00	.00352 47802
.00 C8 00 00	.00305 17578	.00 E8 00 00	.00354 00390
.00 C9 00 00	.00306 70166	.00 E9 00 00	.00355 52978
.00 CA 00 00	.00308 22753	.00 EA 00 00	.00357 05566
.00 CB 00 00	.00309 75341	.00 EB 00 00	.00358 58154
.00 CC 00 00	.00311 27929	.00 EC 00 00	.00360 10742
.00 CD 00 00	.00312 80517	.00 ED 00 00	.00361 63330
.00 CE 00 00	.00314 33105	.00 EE 00 00	.00363 15917
.00 CF 00 00	.00315 85693	.00 EF 00 00	.00364 68505
.00 D0 00 00	.00317 38281	.00 F0 00 00	.00366 21093
.00 D1 00 00	.00318 90869	.00 F1 00 00	.00367 73681
.00 D2 00 00	.00320 43457	.00 F2 00 00	.00369 26269
.00 D3 00 00	.00321 96044	.00 F3 00 00	.00370 78857
.00 D4 00 00	.00323 48632	.00 F4 00 00	.00372 31445
.00 D5 00 00	.00325 01220	.00 F5 00 00	.00373 84033
.00 D6 00 00	.00326 53808	.00 F6 00 00	.00375 36621
.00 D7 00 00	.00328 06396	.00 F7 00 00	.00376 89208
.00 D8 00 00	.00329 58984	.00 F8 00 00	.00378 41796
.00 D9 00 00	.00331 11572	.00 F9 00 00	.00379 94384
.00 DA 00 00	.00332 64160	.00 FA 00 00	.00381 46972
.00 DB 00 00	.00334 16748	.00 FB 00 00	.00382 99560
.00 DC 00 00	.00335 69335	.00 FC 00 00	.00384 52148
.00 DD 00 00	.00337 21923	.00 FD 00 00	.00386 04736
.00 DE 00 00	.00338 74511	.00 FE 00 00	.00387 57324
.00 DF 00 00	.00340 27099	.00 FF 00 00	.00389 09912

Table A-5. Hexadecimal/Decimal Fraction Conversions (Continued)

Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 00	.00000 00000	.00 00 20 00	.00000 19073
.00 00 01 00	.00000 00596	.00 00 21 00	.00000 19669
.00 00 02 00	.00000 01192	.00 00 22 00	.00000 20265
.00 00 03 00	.00000 01788	.00 00 23 00	.00000 20861
.00 00 04 00	.00000 02384	.00 00 24 00	.00000 21457
.00 00 05 00	.00000 02980	.00 00 25 00	.00000 22053
.00 00 06 00	.00000 03576	.00 00 26 00	.00000 22649
.00 00 07 00	.00000 04172	.00 00 27 00	.00000 23245
.00 00 08 00	.00000 04768	.00 00 28 00	.00000 23841
.00 00 09 00	.00000 05364	.00 00 29 00	.00000 24437
.00 00 0A 00	.00000 05960	.00 00 2A 00	.00000 25033
.00 00 0B 00	.00000 06556	.00 00 2B 00	.00000 25629
.00 00 0C 00	.00000 07152	.00 00 2C 00	.00000 26226
.00 00 0D 00	.00000 07748	.00 00 2D 00	.00000 26822
.00 00 0E 00	.00000 08344	.00 00 2E 00	.00000 27418
.00 00 0F 00	.00000 08940	.00 00 2F 00	.00000 28014
.00 00 10 00	.00000 09536	.00 00 30 00	.00000 28610
.00 00 11 00	.00000 10132	.00 00 31 00	.00000 29206
.00 00 12 00	.00000 10728	.00 00 32 00	.00000 29802
.00 00 13 00	.00000 11324	.00 00 33 00	.00000 30398
.00 00 14 00	.00000 11920	.00 00 34 00	.00000 30994
.00 00 15 00	.00000 12516	.00 00 35 00	.00000 31590
.00 00 16 00	.00000 13113	.00 00 36 00	.00000 32186
.00 00 17 00	.00000 13709	.00 00 37 00	.00000 32782
.00 00 18 00	.00000 14305	.00 00 38 00	.00000 33378
.00 00 19 00	.00000 14901	.00 00 39 00	.00000 33974
.00 00 1A 00	.00000 15497	.00 00 3A 00	.00000 34570
.00 00 1B 00	.00000 16093	.00 00 3B 00	.00000 35166
.00 00 1C 00	.00000 16689	.00 00 3C 00	.00000 35762
.00 00 1D 00	.00000 17285	.00 00 3D 00	.00000 36358
.00 00 1E 00	.00000 17881	.00 00 3E 00	.00000 36954
.00 00 1F 00	.00000 18477	.00 00 3F 00	.00000 37550

Table A-5. Hexadecimal/Decimal Fraction Conversions (Continued)

Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 40 00	.00000 38146	.00 00 60 00	.00000 57220
.00 00 41 00	.00000 38743	.00 00 61 00	.00000 57816
.00 00 42 00	.00000 39339	.00 00 62 00	.00000 58412
.00 00 43 00	.00000 39935	.00 00 63 00	.00000 59008
.00 00 44 00	.00000 40531	.00 00 64 00	.00000 59604
.00 00 45 00	.00000 41127	.00 00 65 00	.00000 60200
.00 00 46 00	.00000 41723	.00 00 66 00	.00000 60796
.00 00 47 00	.00000 42319	.00 00 67 00	.00000 61392
.00 00 48 00	.00000 42915	.00 00 68 00	.00000 61988
.00 00 49 00	.00000 43511	.00 00 69 00	.00000 62584
.00 00 4A 00	.00000 44107	.00 00 6A 00	.00000 63180
.00 00 4B 00	.00000 44703	.00 00 6B 00	.00000 63776
.00 00 4C 00	.00000 45299	.00 00 6C 00	.00000 64373
.00 00 4D 00	.00000 45895	.00 00 6D 00	.00000 64969
.00 00 4E 00	.00000 46491	.00 00 6E 00	.00000 65565
.00 00 4F 00	.00000 47087	.00 00 6F 00	.00000 66161
.00 00 50 00	.00000 47683	.00 00 70 00	.00000 66757
.00 00 51 00	.00000 48279	.00 00 71 00	.00000 67353
.00 00 52 00	.00000 48875	.00 00 72 00	.00000 67949
.00 00 53 00	.00000 49471	.00 00 73 00	.00000 68545
.00 00 54 00	.00000 50067	.00 00 74 00	.00000 69141
.00 00 55 00	.00000 50663	.00 00 75 00	.00000 69737
.00 00 56 00	.00000 51259	.00 00 76 00	.00000 70333
.00 00 57 00	.00000 51856	.00 00 77 00	.00000 70929
.00 00 58 00	.00000 52452	.00 00 78 00	.00000 71525
.00 00 59 00	.00000 53048	.00 00 79 00	.00000 72121
.00 00 5A 00	.00000 53644	.00 00 7A 00	.00000 72717
.00 00 5B 00	.00000 54240	.00 00 7B 00	.00000 73313
.00 00 5C 00	.00000 54836	.00 00 7C 00	.00000 73909
.00 00 5D 00	.00000 55432	.00 00 7D 00	.00000 74505
.00 00 5E 00	.00000 56028	.00 00 7E 00	.00000 75101
.00 00 5F 00	.00000 56624	.00 00 7F 00	.00000 75697

Table A-5. Hexadecimal/Decimal Fraction Conversions (Continued)

Hexadécimal	Decimal	Hexadecimal	Decimal
.00 00 80 00	.00000 76293	.00 00 A0 00	.00000 95367
.00 00 81 00	.00000 76889	.00 00 A1 00	.00000 95963
.00 00 82 00	.00000 77486	.00 00 A2 00	.00000 96559
.00 00 83 00	.00000 78082	.00 00 A3 00	.00000 97155
.00 00 84 00	.00000 78678	.00 00 A4 00	.00000 97751
.00 00 85 00	.00000 79274	.00 00 A5 00	.00000 98347
.00 00 86 00	.00000 79870	.00 00 A6 00	.00000 98943
.00 00 87 00	.00000 80466	.00 00 A7 00	.00000 99539
.00 00 88 00	.00000 81062	.00 00 A8 00	.00001 00135
.00 00 89 00	.00000 81658	.00 00 A9 00	.00001 00731
.00 00 8A 00	.00000 82254	.00 00 AA 00	.00001 01327
.00 00 8B 00	.00000 82850	.00 00 AB 00	.00001 01923
.00 00 8C 00	.00000 83446	.00 00 AC 00	.00001 02519
.00 00 8D 00	.00000 84042	.00 00 AD 00	.00001 03116
.00 00 8E 00	.00000 84638	.00 00 AE 00	.00001 03712
.00 00 8F 00	.00000 85234	.00 00 AF 00	.00001 04308
.00 00 90 00	.00000 85830	.00 00 B0 00	.00001 04904
.00 00 91 00	.00000 86426	.00 00 B1 00	.00001 05500
.00 00 92 00	.00000 87022	.00 00 B2 00	.00001 06096
.00 00 93 00	.00000 87618	.00 00 B3 00	.00001 06692
.00 00 94 00	.00000 88214	.00 00 B4 00	.00001 07288
.00 00 95 00	.00000 88810	.00 00 B5 00	.00001 07884
.00 00 96 00	.00000 89406	.00 00 B6 00	.00001 08480
.00 00 97 00	.00000 90003	.00 00 B7 00	.00001 09076
.00 00 98 00	.00000 90599	.00 00 B8 00	.00001 09672
.00 00 99 00	.00000 91195	.00 00 B9 00	.00001 10268
.00 00 9A 00	.00000 91791	.00 00 BA 00	.00001 10864
.00 00 9B 00	.00000 92387	.00 00 BB 00	.00001 11460
.00 00 9C 00	.00000 92983	.00 00 BC 00	.00001 12056
.00 00 9D 00	.00000 93579	.00 00 BD 00	.00001 12652
.00 00 9E 00	.00000 94175	.00 00 BE 00	.00001 13248
.00 00 9F 00	.00000 94771	.00 00 BF 00	.00001 13844

Table A-5. Hexadecimal/Decimal Fraction Conversions (Continued)

Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 C0 00	.00001 14440	.00 00 E0 00	.00001 33514
.00 00 C1 00	.00001 15036	.00 00 E1 00	.00001 34110
.00 00 C2 00	.00001 15633	.00 00 E2 00	.00001 34706
.00 00 C3 00	.00001 16229	.00 00 E3 00	.00001 35302
.00 00 C4 00	.00001 16825	.00 00 E4 00	.00001 35898
.00 00 C5 00	.00001 17421	.00 00 E5 00	.00001 36494
.00 00 C6 00	.00001 18017	.00 00 E6 00	.00001 37090
.00 00 C7 00	.00001 18613	.00 00 E7 00	.00001 37686
.00 00 C8 00	.00001 19209	.00 00 E8 00	.00001 38282
.00 00 C9 00	.00001 19805	.00 00 E9 00	.00001 38878
.00 00 CA 00	.00001 20401	.00 00 EA 00	.00001 39474
.00 00 CB 00	.00001 20997	.00 00 EB 00	.00001 40070
.00 00 CC 00	.00001 21593	.00 00 EC 00	.00001 40666
.00 00 CD 00	.00001 22189	.00 00 ED 00	.00001 41263
.00 00 CE 00	.00001 22785	.00 00 EE 00	.00001 41859
.00 00 CF 00	.00001 23381	.00 00 EF 00	.00001 42455
.00 00 D0 00	.00001 23977	.00 00 F0 00	.00001 43051
.00 00 D1 00	.00001 24573	.00 00 F1 00	.00001 43647
.00 00 D2 00	.00001 25169	.00 00 F2 00	.00001 44243
.00 00 D3 00	.00001 25765	.00 00 F3 00	.00001 44839
.00 00 D4 00	.00001 26361	.00 00 F4 00	.00001 45435
.00 00 D5 00	.00001 26957	.00 00 F5 00	.00001 46031
.00 00 D6 00	.00001 27553	.00 00 F6 00	.00001 46627
.00 00 D7 00	.00001 28149	.00 00 F7 00	.00001 47223
.00 00 D8 00	.00001 28746	.00 00 F8 00	.00001 47819
.00 00 D9 00	.00001 29342	.00 00 F9 00	.00001 48415
.00 00 DA 00	.00001 29938	.00 00 FA 00	.00001 49011
.00 00 DB 00	.00001 30534	.00 00 FB 00	.00001 49607
.00 00 DC 00	.00001 31130	.00 00 FC 00	.00001 50203
.00 00 DD 00	.00001 31726	.00 00 FD 00	.00001 50799
.00 00 DE 00	.00001 32322	.00 00 FE 00	.00001 51395
.00 00 DF 00	.00001 32918	.00 00 FF 00	.00001 51991

Table A-5. Hexadecimal/Decimal Fraction Conversions (Continued)

Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 00	.00000 00000	.00 00 00 20	.00000 00074
.00 00 00 01	.00000 00002	.00 00 00 21	.00000 00076
.00 00 00 02	.00000 00004	.00 00 00 22	.00000 00079
.00 00 00 03	.00000 00006	.00 00 00 23	.00000 00081
.00 00 00 04	.00000 00009	.00 00 00 24	.00000 00083
.00 00 00 05	.00000 00011	.00 00 00 25	.00000 00086
.00 00 00 06	.00000 00013	.00 00 00 26	.00000 00088
.00 00 00 07	.00000 00016	.00 00 00 27	.00000 00090
.00 00 00 08	.00000 00018	.00 00 00 28	.00000 00093
.00 00 00 09	.00000 00020	.00 00 00 29	.00000 00095
.00 00 00 0A	.00000 00023	.00 00 00 2A	.00000 00097
.00 00 00 0B	.00000 00025	.00 00 00 2B	.00000 00100
.00 00 00 0C	.00000 00027	.00 00 00 2C	.00000 00102
.00 00 00 0D	.00000 00030	.00 00 00 2D	.00000 00104
.00 00 00 0E	.00000 00032	.00 00 00 2E	.00000 00107
.00 00 00 0F	.00000 00034	.00 00 00 2F	.00000 00109
.00 00 00 10	.00000 00037	.00 00 00 30	.00000 00111
.00 00 00 11	.00000 00039	.00 00 00 31	.00000 00114
.00 00 00 12	.00000 00041	.00 00 00 32	.00000 00116
.00 00 00 13	.00000 00044	.00 00 00 33	.00000 00118
.00 00 00 14	.00000 00046	.00 00 00 34	.00000 00121
.00 00 00 15	.00000 00048	.00 00 00 35	.00000 00123
.00 00 00 16	.00000 00051	.00 00 00 36	.00000 00125
.00 00 00 17	.00000 00053	.00 00 00 37	.00000 00128
.00 00 00 18	.00000 00055	.00 00 00 38	.00000 00130
.00 00 00 19	.00000 00058	.00 00 00 39	.00000 00132
.00 00 00 1A	.00000 00060	.00 00 00 3A	.00000 00135
.00 00 00 1B	.00000 00062	.00 00 00 3B	.00000 00137
.00 00 00 1C	.00000 00065	.00 00 00 3C	.00000 00139
.00 00 00 1D	.00000 00067	.00 00 00 3D	.00000 00142
.00 00 00 1E	.00000 00069	.00 00 00 3E	.00000 00144
.00 00 00 1F	.00000 00072	.00 00 00 3F	.00000 00146

Table A-5. Hexadecimal/Decimal Fraction Conversions (Continued)

Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 40	.00000 00149	.00 00 00 60	.00000 00223
.00 00 00 41	.00000 00151	.00 00 00 61	.00000 00225
.00 00 00 42	.00000 00153	.00 00 00 62	.00000 00228
.00 00 00 43	.00000 00155	.00 00 00 63	.00000 00230
.00 00 00 44	.00000 00158	.00 00 00 64	.00000 00232
.00 00 00 45	.00000 00160	.00 00 00 65	.00000 00235
.00 00 00 46	.00000 00162	.00 00 00 66	.00000 00237
.00 00 00 47	.00000 00165	.00 00 00 67	.00000 00239
.00 00 00 48	.00000 00167	.00 00 00 68	.00000 00242
.00 00 00 49	.00000 00169	.00 00 00 69	.00000 00244
.00 00 00 4A	.00000 00172	.00 00 00 6A	.00000 00246
.00 00 00 4B	.00000 00174	.00 00 00 6B	.00000 00249
.00 00 00 4C	.00000 00176	.00 00 00 6C	.00000 00251
.00 00 00 4D	.00000 00179	.00 00 00 6D	.00000 00253
.00 00 00 4E	.00000 00181	.00 00 00 6E	.00000 00256
.00 00 00 4F	.00000 00183	.00 00 00 6F	.00000 00258
.00 00 00 50	.00000 00186	.00 00 00 70	.00000 00260
.00 00 00 51	.00000 00188	.00 00 00 71	.00000 00263
.00 00 00 52	.00000 00190	.00 00 00 72	.00000 00265
.00 00 00 53	.00000 00193	.00 00 00 73	.00000 00267
.00 00 00 54	.00000 00195	.00 00 00 74	.00000 00270
.00 00 00 55	.00000 00197	.00 00 00 75	.00000 00272
.00 00 00 56	.00000 00200	.00 00 00 76	.00000 00274
.00 00 00 57	.00000 00202	.00 00 00 77	.00000 00277
.00 00 00 58	.00000 00204	.00 00 00 78	.00000 00279
.00 00 00 59	.00000 00207	.00 00 00 79	.00000 00281
.00 00 00 5A	.00000 00209	.00 00 00 7A	.00000 00284
.00 00 00 5B	.00000 00211	.00 00 00 7B	.00000 00286
.00 00 00 5C	.00000 00214	.00 00 00 7C	.00000 00288
.00 00 00 5D	.00000 00216	.00 00 00 7D	.00000 00291
.00 00 00 5E	.00000 00218	.00 00 00 7E	.00000 00293
.00 00 00 5F	.00000 00221	.00 00 00 7F	.00000 00295

Table A-5. Hexadecimal/Decimal Fraction Conversions (Continued)

Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 80	.00000 00298	.00 00 00 A0	.00000 00372
.00 00 00 81	.00000 00300	.00 00 00 A1	.00000 00374
.00 00 00 82	.00000 00302	.00 00 00 A2	.00000 00377
.00 00 00 83	.00000 00305	.00 00 00 A3	.00000 00379
.00 00 00 84	.00000 00307	.00 00 00 A4	.00000 00381
.00 00 00 85	.00000 00309	.00 00 00 A5	.00000 00384
.00 00 00 86	.00000 00311	.00 00 00 A6	.00000 00386
.00 00 00 87	.00000 00314	.00 00 00 A7	.00000 00388
.00 00 00 88	.00000 00316	.00 00 00 A8	.00000 00391
.00 00 00 89	.00000 00318	.00 00 00 A9	.00000 00393
.00 00 00 8A	.00000 00321	.00 00 00 AA	.00000 00395
.00 00 00 8B	.00000 00329	.00 00 00 AB	.00000 00398
.00 00 00 8C	.00000 00325	.00 00 00 AC	.00000 00400
.00 00 00 8D	.00000 00328	.00 00 00 AD	.00000 00402
.00 00 00 8E	.00000 00330	.00 00 00 AE	.00000 00405
.00 00 00 8F	.00000 00332	.00 00 00 AF	.00000 00407
.00 00 00 90	.00000 00335	.00 00 00 B0	.00000 00409
.00 00 00 91	.00000 00337	.00 00 00 B1	.00000 00412
.00 00 00 92	.00000 00339	.00 00 00 B2	.00000 00414
.00 00 00 93	.00000 00342	.00 00 00 B3	.00000 00416
.00 00 00 94	.00000 00344	.00 00 00 B4	.00000 00419
.00 00 00 95	.00000 00346	.00 00 00 B5	.00000 00421
.00 00 00 96	.00000 00349	.00 00 00 B6	.00000 00423
.00 00 00 97	.00000 00351	.00 00 00 B7	.00000 00426
.00 00 00 98	.00000 00353	.00 00 00 B8	.00000 00428
.00 00 00 99	.00000 00356	.00 00 00 B9	.00000 00430
.00 00 00 9A	.00000 00358	.00 00 00 BA	.00000 00433
.00 00 00 9B	.00000 00360	.00 00 00 BB	.00000 00435
.00 00 00 9C	.00000 00363	.00 00 00 BC	.00000 00437
.00 00 00 9D	.00000 00365	.00 00 00 BD	.00000 00440
.00 00 00 9E	.00000 00367	.00 00 00 BE	.00000 00442
.00 00 00 9F	.00000 00370	.00 00 00 BF	.00000 00444

Table A-5. Hexadecimal/Decimal Fraction Conversions (Continued)

Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 C0	.00000 00447	.00 00 00 E0	.00000 00521
.00 00 00 C1	.00000 00449	.00 00 00 E1	.00000 00523
.00 00 00 C2	.00000 00451	.00 00 00 E2	.00000 00526
.00 00 00 C3	.00000 00454	.00 00 00 E3	.00000 00528
.00 00 00 C4	.00000 00456	.00 00 00 E4	.00000 00530
.00 00 00 C5	.00000 00458	.00 00 00 E5	.00000 00533
.00 00 00 C6	.00000 00461	.00 00 00 E6	.00000 00535
.00 00 00 C7	.00000 00463	.00 00 00 E7	.00000 00537
.00 00 00 C8	.00000 00465	.00 00 00 E8	.00000 00540
.00 00 00 C9	.00000 00467	.00 00 00 E9	.00000 00542
.00 00 00 CA	.00000 00470	.00 00 00 EA	.00000 00544
.00 00 00 CB	.00000 00472	.00 00 00 EB	.00000 00547
.00 00 00 CC	.00000 00474	.00 00 00 EC	.00000 00549
.00 00 00 CD	.00000 00477	.00 00 00 ED	.00000 00551
.00 00 00 CE	.00000 00479	.00 00 00 EE	.00000 00554
.00 00 00 CF	.00000 00481	.00 00 00 EF	.00000 00556
.00 00 00 D0	.00000 00484	.00 00 00 F0	.00000 00558
.00 00 00 D1	.00000 00486	.00 00 00 F1	.00000 00561
.00 00 00 D2	.00000 00488	.00 00 00 F2	.00000 00563
.00 00 00 D3	.00000 00491	.00 00 00 F3	.00000 00565
.00 00 00 D4	.00000 00493	.00 00 00 F4	.00000 00568
.00 00 00 D5	.00000 00495	.00 00 00 F5	.00000 00570
.00 00 00 D6	.00000 00498	.00 00 00 F6	.00000 00572
.00 00 00 D7	.00000 00500	.00 00 00 F7	.00000 00575
.00 00 00 D8	.00000 00502	.00 00 00 F8	.00000 00577
.00 00 00 D9	.00000 00505	.00 00 00 F9	.00000 00579
.00 00 00 DA	.00000 00507	.00 00 00 FA	.00000 00582
.00 00 00 DB	.00000 00509	.00 00 00 FB	.00000 00584
.00 00 00 DC	.00000 00512	.00 00 00 FC	.00000 00586
.00 00 00 DD	.00000 00514	.00 00 00 FD	.00000 00589
.00 00 00 DE	.00000 00516	.00 00 00 FE	.00000 00591
.00 00 00 DF	.00000 00519	.00 00 00 FF	.00000 00593

Table A-6. Hexadecimal Arithmetic

Addition

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10
2	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11
3	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12
4	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
5	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14
6	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15
7	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16
8	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17
9	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18
A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19
B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A
C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	A	1B
D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Multiplication

1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
3	06	09	0C	0F	12	15	18	1B	1E	21	24	27	2A	2D
4	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0A	0F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

Table A-7. Mathematical Constants

Constant	Decimal Value	Hexadecimal Value
π	3.14159 26535 89793	3.243F 6A89
π^{-1}	0.31830 98861 83790	0.517C C1B7
$\sqrt{\pi}$	1.77245 38509 05516	1.C5BF 891C
$\ln \pi$	1.14472 98858 49400	1.250D 048F
e	2.71828 18284 59045	2.B7E1 5163
e^{-1}	0.36787 94411 71442	0.5E2D 58D9
$\sqrt{e}$	1.64872 12707 00128	1.A612 98E2
$\log_{10} e$	0.43429 44819 03252	0.6F2D EC55
$\log_2 e$	1.44269 50408 88963	1.7154 7653
γ	0.57721 56649 01533	0.93C4 67E4
$\ln \gamma$	-0.54953 93129 81645	-0.8CAE 9BC1
$\sqrt{2}$	1.41421 35623 73095	1.6A09 E668
$\ln 2$	0.69314 71805 59945	0.B172 17F8
$\log_{10} 2$	0.30102 99956 63981	0.4D10 4D42
$\sqrt{10}$	3.16227 76601 68379	3.298B 075C
$\ln 10$	2.30258 50929 94046	2.4D76 3777

Table A-8. Powers of 16, Base 10

16^n	n	16^{-n}
1	0	0.10000 00000 00000 00000 x 10^0
16	1	0.62500 00000 00000 00000 x 10^{-1}
256	2	0.39062 50000 00000 00000 x 10^{-2}
4 096	3	0.24414 06250 00000 00000 x 10^{-3}
65 536	4	0.15258 78906 25000 00000 x 10^{-4}
1 048 576	5	0.95367 43164 06250 00000 x 10^{-6}
16 777 216	6	0.59604 64477 53906 25000 x 10^{-7}
268 435 456	7	0.37252 90298 46191 40625 x 10^{-8}
4 294 967 296	8	0.23283 06436 53869 62891 x 10^{-9}
68 719 476 736	9	0.14551 9152 83668 51807 x 10^{-10}
1 099 511 627 776	10	0.90949 47017 72928 23792 x 10^{-12}
17 592 186 044 416	11	0.56843 41886 08080 14870 x 10^{-13}
281 474 976 710 656	12	0.35527 13678 80050 09294 x 10^{-14}
4 503 599 627 370 496	13	0.22204 46049 25031 30808 x 10^{-15}
72 057 594 037 927 936	14	0.13877 78780 78144 56755 x 10^{-16}
1 152 921 504 606 846 976	15	0.86736 17379 88403 54721 x 10^{-18}

Table A-9. Powers of 10, Base 16

10^n	n	10^{-n}
1	0	1.0000 0000 0000 0000
A	1	0.1999 9999 9999 999A
64	2	0.28F5 C28F 5C28 F5C3 $\times 16^{-1}$
3E8	3	0.4189 374B C6A7 EF9E $\times 16^{-2}$
2710	4	0.68DB 8BAC 710C B296 $\times 16^{-3}$
1 86A0	5	0.A7C5 AC47 1B47 8423 $\times 16^{-4}$
F 4240	6	0.10C6 F7A0 B5ED 8D37 $\times 16^{-4}$
98 9680	7	0.1AD7 F29A BCAF 4858 $\times 16^{-5}$
5F5 E100	8	0.2AF3 1DC4 6118 73BF $\times 16^{-6}$
3B9A CA00	9	0.44B8 2FA0 9B5A 52CC $\times 16^{-7}$
2 540B E400	10	0.6DF3 7F67 5EF6 EADF $\times 16^{-8}$
17 4876 E800	11	0.AFEB FF0B CB24 AAFB $\times 16^{-9}$
E8 D4A5 1000	12	0.1197 9981 2DEA 1119 $\times 16^{-9}$
918 4E72 A000	13	0.1C25 C268 4976 81C2 $\times 16^{-10}$
5AF3 107A 4000	14	0.2D09 370D 4257 3604 $\times 16^{-11}$
3 8D7E A4C6 8000	15	0.480E BE7B 9D58 566D $\times 16^{-12}$
23 8652 6FC1 0000	16	0.734A CA5F 6226 F0AE $\times 16^{-13}$
163 4578 5D8A 0000	17	0.B877 AA32 36A4 B449 $\times 16^{-14}$
DE0 B6B3 A764 0000	18	0.1272 5DD1 D243 ABA1 $\times 16^{-14}$
8AC7 2304 89E8 0000	19	0.1D83 C94F B6D2 AC35 $\times 16^{-15}$

Standard Subroutine Running Times

Routine Symbol and Name		Storage — bytes		Time-Cycles		Average Time Milliseconds	Routine Label
		Routine	Temp	Minimum	Maximum		
FIXED POINT							
MU	Multiply — 2 byte	107	4	101	154	.19	U
DI	Divide — 2 byte	133	10	261	274	.40	D
DB	Decimal to Binary	98	8	106	880	.70	K
BD	Binary to Decimal	154	10	154	2084	2.00	B
FLOATING POINT							
RX	Load X Immediate	20	2	35	35	.05	X
SE	Separate Exponent	28	3	34	42	.05	F
FC	Construct Floating Point	28	3	36	42	.06	C
FN	Normalize	70	6	82	261	.26	N
IF	Integer to Float	54	7	74	319	.35	J
FI	Float to Integer	70	7	71	346	.30	I
FA	Add	144	8	336	634	.74	A
FS	Subtract	32	4	388	687	.82	S
FM	Multiply	128	10	909	996	1.42	M
FD	Divide	146	14	1125	1236	1.76	V
FP	Fractional Part			127	648	.60	P
IP	Integer Part	110	2	124	704	.60	P
SQ	Square Root	105	12	5090	5455	7.83	Q
EX	Exponential e ^X	198	10	9440	9950	14.54	E
LN	Natural Log.	173	10	----	----	15.82	L
FE	Exponentiation A**B	23	0	----	----	31.81	Z
QF	Decimal to Float	163	6	----	----	18.15	W
FQ	Float to Decimal	138	8	----	----	22.80	R
SI	Sine	206	10	13250	16100	19.95	O
CO	Cosine	27	0	13700	16500	20.50	O
AT	Arctangent	203	6	----	----	27.00	T

NOTES

1. One cycle equals 1.5 microseconds
2. Times include the time of all other routines called during execution.

Binary Object Tape Format

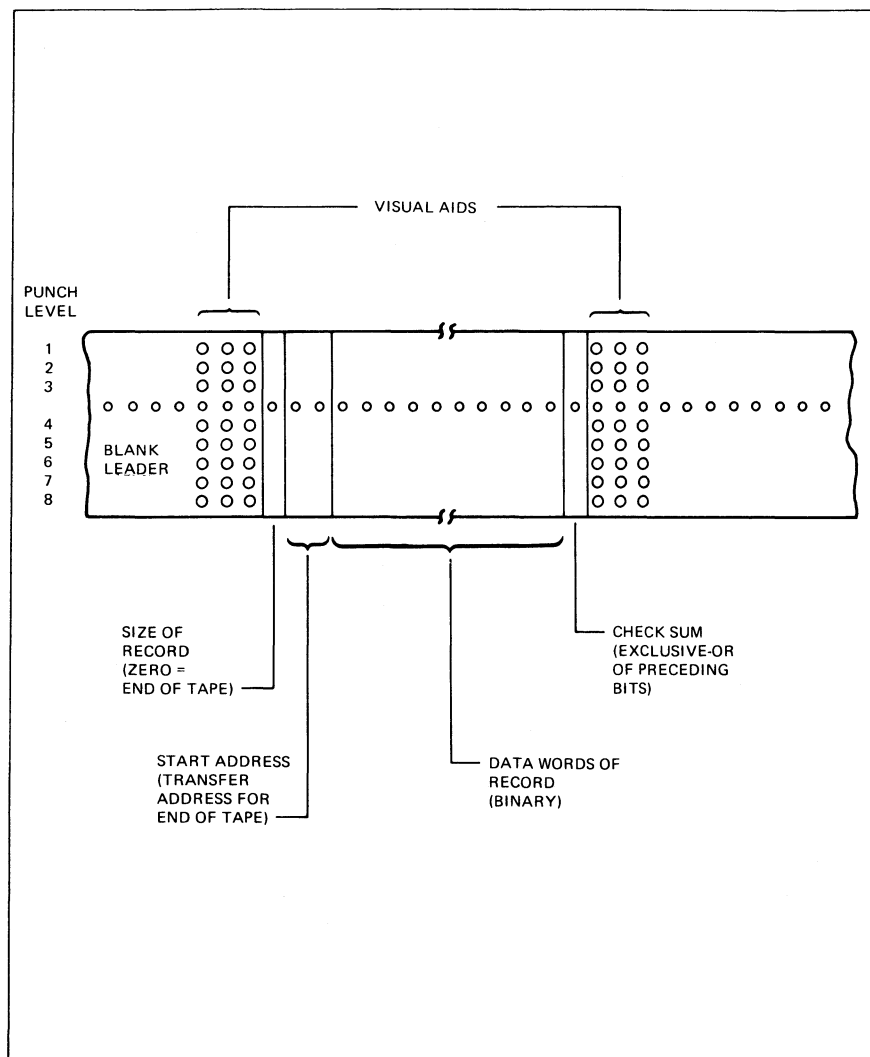
All statements of a Varian 520/i object tape are punched in the binary equivalent of their hexadecimal code. For example, the code for the branch-and-mark instruction (BRM) is 38. This is punched in one line as 0011 1000.

The physical format of an object tape record is shown on the opposite page. The drawing shows the six parts of each tape record. These are:

1. Blank leader that is positioned over the tape read station prior to reading tape.
2. A three-line visual aid just before and just after the program record. Each line of the visual aid is fully punched.
3. Size-of-record is the first line after the visual aid and indicates, in binary format, the number of data words in the record.
4. Following the size-of-record line is the start address on two lines. This 16-bit address identifies any location in the maximum memory as the starting location to store data words of the object record.

When the size-of-record line is zero, the two address lines indicate a transfer address for the Varian 520/i program counter. The contents of the transfer location may be the first instruction of the program that was just loaded or of another program. Alternately, the transfer location may contain a halt command or other programming aid.

5. Data words that comprise the object record follow the start address. The data words are sequentially stored in memory beginning at the start address.
6. A check sum follows immediately after the data words. Each bit of the check sum is the exclusive-OR combination of the corresponding bits of all preceding data in the record including data words, address and record length.



Standard Character Codes

Character	ASCII Binary	ASCII Hexadecimal	Hollerith
@	11000000	C0	0-2-8
A	11000001	C1	12-1
B	11000010	C2	12-2
C	11000011	C3	12-3
D	11000100	C4	12-4
E	11000101	C5	12-5
F	11000110	C6	12-6
G	11000111	C7	12-7
H	11001000	C8	12-8
I	11001001	C9	12-9
J	11001010	CA	11-1
K	11001011	CB	11-2
L	11001100	CC	11-3
M	11001101	CD	11-4
N	11001110	CE	11-5
O	11001111	CF	11-6
P	11010000	D0	11-7
Q	11010001	D1	11-8
R	11010010	D2	11-9
S	11010011	D3	0-2
T	11010100	D4	0-3
U	11010101	D5	0-4
V	11010110	D6	0-5
W	11010111	D7	0-6
X	11011000	D8	0-7
Y	11011001	D9	0-8
Z	11011010	DA	0-9
[	11011011	DB	12-5-8
\	11011100	DC	0-6-8
]	11011101	DD	11-5-8
↑	11011110	DE	7-8
←	11011111	DF	2-8
blank	10100000	A0	no punch

Character	ASCII Binary	ASCII Hexadecimal	Hollerith
'	10100001	A1	11-2-8
''	10100010	A2	0-5-8
#	10100011	A3	0-7-8
\$	10100100	A4	11-3-8
%	10100101	A5	11-7-8
&	10100110	A6	12-7-8
!	10100111	A7	4-8
(	10101000	A8	0-4-8
)	10101001	A9	12-4-8
*	10101010	AA	11-4-8
+	10101011	AB	12
,	10101100	AC	0-3-8
-	10101101	AD	11
.	10101110	AE	12-3-8
/	10101111	AF	0-1
0	10110000	B0	0
1	10110001	B1	1
2	10110010	B2	2
3	10110011	B3	3
4	10110100	B4	4
5	10110101	B5	5
6	10110110	B6	6
7	10110111	B7	7
8	10111000	B8	8
9	10111001	B9	9
:	10111010	BA	5-8
;	10111011	BB	11-6-8
<	10111100	BC	12-6-8
=	10111101	BD	3-8
>	10111110	BE	6-8
?	10111111	BF	12-2-8

Assembly System Error Characters

Character	Definition
O	Invalid operation mnemonic
U	Undefined symbol
M	Multiply defined symbol
P	Invalid operand precision
N	Invalid numerical operand
I	Invalid operation code addition
A	Addressing error

Fixed addresses have been assigned to the peripherals that are most commonly used with the Varian 520/i computer. These are tabulated below.

Address Assigned	Peripheral or Option
00	Computer internal
01	ASR-33/35 teletype
02	Punched-tape system
03	Real-time clock
04	DMA multiplexer
05	Card reader
06	201 modem
08	Magnetic tape
0A-0D	Magnetic tape controller (620/i-30)
0A-0D	Magnetic tape controller (620/i-31)
0E	Rotating memory and controller (620/i-40, 41, 42 and 43)
18	Card reader controller (620/i-51)
1A	Cal Comp 565 plotter (620/i-72)
ID	High-speed printer
1F	Punched-tape system (620/i-50, 51 and 52)

Index of Symbolic Terms

Term	Definition
()	Contents of a register or memory location.
$\overline{(\)}$	Inverse or one's complement of the contents of a register or memory location.
$\overline{x}$	Inverse or one's complement of expression x.
+	Arithmetic sum (where $0 + 0 = 0$, $0 + 1 = 1$, $1 + 0 = 1$ and $1 + 1 = 10$).
—	Arithmetic difference (where $0 - 0 = 0$, $0 - 1 = -1$, $1 - 0 = 1$ and $1 - 1 = 0$).
$\cap$	Logical AND (where $0 \cap 0 = 0$, $0 \cap 1 = 0$, $1 \cap 0 = 0$ and $1 \cap 1 = 1$).
$\cup$	Logical OR (where $0 \cup 0 = 0$, $0 \cup 1 = 1$, $1 \cup 0 = 1$ and $1 \cup 1 = 1$).
$\oplus$	Logical exclusive OR (where $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$ and $1 \oplus 1 = 0$).
(A) (P)	Contents of A are replaced by the contents of P.
OP	Current operand-precision in bytes (1, 2, 3 or 4).
OPC	Two-bit operand precision counter in the current environment.
OVC	Overflow indicator in the current environment.
OPN	Two-bit operand-precision counter in the noncurrent environment.
OVN	Overflow indicator in the noncurrent environment.
Z	16-bit pseudoregister containing all binary ZEROS.
(Acc)	Contents of the entire accumulator in the current environment with an operand precision equal to that of current environment (may be one, two, three or four bytes).
A	Most-significant 16 bits of the accumulator in the current environment.

Term	Definition
C	Least-significant 16 bits of the accumulator in the current environment.
X	16-bit index register in the current environment.
P	16-bit program counter in the current environment.
B	Most-significant 16 bits of the accumulator in the noncurrent environment.
D	Least-significant 16 bits of the accumulator in the noncurrent environment.
Y	16-bit index register in the noncurrent environment.
Q	16-bit program counter in the noncurrent environment.
F	16-bit input/output register in the peripheral adapter.
y	Operand address in a memory-reference instruction.
s	Source register in a register-change instruction.
d	Destination register in a register-change instruction.
o	Order code in an input/output instruction.
a	Device address in an input/output instruction.
m	Addressing mode of an instruction.
[]	Brackets enclose explanatory comments relating to the expression immediately preceding the brackets. For example, $y + 2$ [if $\text{AccMSBit} = 1$] means that two is added to y if the most significant bit of the current accumulator contains a binary ONE.

MEMORY-REFERENCE INSTRUCTION

LOD Load accumulator $(Acc) \leftarrow (y)$

7 6 5 4 | 3 2 1 0

0	1	0	m	y
---	---	---	---	---

First Byte

7 6 5 4 | 3 2 1 0

y					
---	--	--	--	--	--

Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	3	4	5	6

The contents of the memory location(s) specified by the effective address are placed in the accumulator. The contents of memory are unaffected.

STO Store accumulator $(y) \leftarrow (Acc)$

7 6 5 4 | 3 2 1 0

0	1	1	m	y
---	---	---	---	---

First Byte

7 6 5 4 | 3 2 1 0

y					
---	--	--	--	--	--

Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	3	4	5	6

The contents of the accumulator are placed in the memory location(s) specified by the effective address. The accumulator is unaffected.

SUM Add to accumulator $(Acc) \leftarrow (Acc) + (y)$

7 6 5 4 | 3 2 1 0

1	0	0	m	y
---	---	---	---	---

First Byte

7 6 5 4 | 3 2 1 0

y					
---	--	--	--	--	--

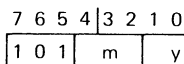
Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	3	4	5	6

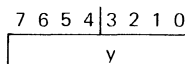
The arithmetic sum of the contents of the accumulator and the contents of the memory location(s) specified by the effective address are placed in the accumulator. The contents of memory are unaffected. The overflow indicator is set if the result is less than -2^{AL-1} or greater than $2^{AL-1} - 1$, where AL is the accumulator length in bits.

SUB Subtract from accumulator

$$(Acc) \leftarrow (Acc) - (y)$$



First Byte



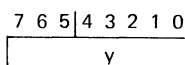
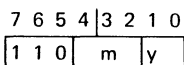
Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	3	4	5	6

The arithmetic difference between the contents of the accumulator and the contents of the memory location(s) specified by the effective address are placed in the accumulator. The contents of memory are unaffected. The overflow indicator is set if the result is less than -2^{AL-1} or greater than $2^{AL-1} - 1$, where AL is the accumulator length in bits.

AND AND to accumulator

$$(Acc) \leftarrow (Acc) \cap (y)$$

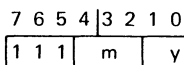


Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	3	4	5	6

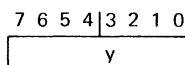
The logical product of the contents of the accumulator and the contents of the memory location(s) specified by the effective address are placed in the accumulator. The contents of memory are unaffected.

BRU Branch unconditionally

$$(P) \leftarrow y$$



First Byte



Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	2	2	2	2

The effective address (after any address modification) is placed in the current program counter.

Non-Memory One-Byte Instructions, Register-Operate

NON-MEMORY-REFERENCE ONE-BYTE INSTRUCTIONS

Register-Operate Instructions

CLA Clear accumulator

$(Acc) \leftarrow 0$

7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	2	2	2

The contents of the current accumulator are set to zero. The accumulator is affected only to the set precision.

CMA Complement accumulator

$(Acc) \leftarrow (Acc)$

7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	2	2	2

The contents of the current accumulator are inverted (one's complement). The accumulator is affected only to the set precision.

IAR Increment accumulator

$(Acc) \leftarrow (Acc) + 1$

7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	0

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	2	2	2

The contents of the current accumulator are incremented by one. The accumulator is affected only to the set precision. If the result is greater than $2^{AL}-1$, where AL is the accumulator length in bits, the overflow indicator is set.

CIA Complement and increment accumulator

$(Acc) \leftarrow (Acc) + 1$

7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	2	2	2

The contents of the current accumulator are inverted and incremented (two's complement). The accumulator is affected only to the set precision. If the result is less than $-2^{AL}-1$, where AL is the accumulator length in bits, the overflow indicator is set.

**IXO Increment index register
by the operand precision.**

$(X) \leftarrow (X) + (OPC)$

7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	2	2	2	2

The current operand precision (in bytes) is added to the contents of the current index register.

Non-Memory One-Byte Instructions, Control

Control Instruction

SP1 Set operand precision to 1 byte

OP = 1 byte

7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	0

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	1	1	1

The accumulator and operand length is adjusted to eight bits.

SP2 Set operand precision to 2 bytes

OP = 2 bytes

7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	1

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	1	1	1

The accumulator and operand length is adjusted to 16 bits.

SP3 Set operand precision to 3 bytes

OP = 3 bytes

7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	0

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	1	1	1

The accumulator and operand length is adjusted to 24 bits.

SP4 Set operand precision to 4 bytes

OP = 4 bytes

7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	1

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	1	1	1

The accumulator and operand length is adjusted to 32 bits.

HLT Halt

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	1	1	1

The machine halts with the program counter containing the next location in sequence.

NOP **No operation**

7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	0

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	1	1	1

No execution activity takes place during this instruction.

SOF **Set overflow indicator**

7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	0

(OVC) $\leftarrow$ 1

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	1	1	1

The current overflow indicator is set.

ROF **Reset overflow indicator**

7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	1

(OVC) $\leftarrow$ 0

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	1	1	1

The current overflow indicator is reset.

Non-Memory One-Byte Instructions, Skip

Skip Instructions

SAN Skip if accumulator is negative

7 6 5 4 | 3 2 1 0

$(P) \leftarrow (P) + 1 + 2$ [if (Acc) < 0]

0 0 0 0 | 0 1 1 0

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	1	1	1

If the contents of the current accumulator are less than zero, the next two bytes in sequence are skipped. The accumulator is tested only to the set precision.

SANN Skip if accumulator is not negative

7 6 5 4 | 3 2 1 0

$(P) \leftarrow (P) + 1 + 2$ [if (Acc) ≥ 0]

0 0 0 0 | 0 1 1 1

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	1	1	1

If the contents of the current accumulator are not less than zero, the next two bytes in sequence are skipped. The accumulator is tested only to the set precision.

SAZ Skip if accumulator is zero

7 6 5 4 | 3 2 1 0

$(P) \leftarrow (P) + 1 + 2$ [if (Acc) = 0]

0 0 0 0 | 1 0 0 0

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	2	2	2

If the contents of the current accumulator are zero, the next two bytes in sequence are skipped. The accumulator is tested only to the set precision.

SAZN Skip if accumulator is not zero

7 6 5 4 | 3 2 1 0

$(P) \leftarrow (P) + 1 + 2$ [if (Acc) $\neq 0$]

0 0 0 0 | 1 0 0 1

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	2	2	2

If the contents of the current accumulator are not zero, the next two bytes in sequence are skipped. The accumulator is tested only to the set precision.

SXZ Skip if index register is zero

$(P) \leftarrow (P) + 1 + 2$ [if $(X) = 0$]

7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	2	2	2	2

If the contents of the current index register are zero the next two bytes in sequence are skipped.

SXZN Skip if index register is not zero

$(P) \leftarrow (P) + 1 + 2$ [if $(X) \neq 0$]

7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	2	2	2	2

If the contents of the current index register are not zero, the next two bytes in sequence are skipped.

SOV Skip on accumulator overflow

$(P) \leftarrow (P) + 1 + 2$ [if $(OVC) = 1$]

$(OVC) \leftarrow 0$

7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	1	1	1

If the overflow indicator of the current accumulator has been set by a previous operation, the next two bytes in sequence are skipped. The current overflow indicator is reset regardless of the test outcome.

SOVN Skip on no accumulator overflow

$(P) \leftarrow (P) + 1 + 2$ [if $(OVC) = 0$]

7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	1	1	1

If the overflow indicator of the current accumulator has not been set by a previous operation, the next two bytes in sequence are skipped. The current overflow indicator is reset regardless of the test outcome.

non-memory one-byte instructions, skip

SOD Skip if accumulator is odd

7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0

$(P) \leftarrow (P) + 1 + 2$ [if $(\text{Acc}_{\text{LSBit}}) = 1$]

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	1	1	1

If the least-significant bit of the current accumulator is a ONE, the next two bytes in sequence are skipped.

SODN Skip if accumulator is not odd

7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1

$(P) \leftarrow (P) + 1 + 2$ [if $(\text{Acc}_{\text{LSBit}}) = 0$]

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	1	1	1

If the least-significant bit of the current accumulator is not a ONE, the next two bytes in sequence are skipped.

Shift Instructions

SR1 Shift right 1 bit

$$(Acc_n) \leftarrow (Acc_{n+1})$$

$$(Acc_{MSBit}) \leftarrow 0$$

7 6 5 4 | 3 2 1 0

0 0 0 1 | 1 0 0 0

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	2	2	2

The current accumulator is shifted one bit to the right. ZERO is shifted into the most-significant bit. The accumulator is affected only to the set precision.

SR8 Shift right 8 bits (1 byte)

$$(Acc_n) \leftarrow (Acc_{n+8})$$

$$(Acc_{MSByte}) \leftarrow 0$$

7 6 5 4 | 3 2 1 0

0 0 0 1 | 1 0 0 1

First Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	2	2	2

The current accumulator is shifted eight bits to the right. ZEROs are shifted into the most-significant byte. The accumulator is affected only to the set precision.

RR1 Rotate right 1 bit

$$(Acc_n) \leftarrow (Acc_{n+1})$$

$$(Acc_{MSBit}) \leftarrow (Acc_{LSBit})$$

7 6 5 4 | 3 2 1 0

0 0 0 1 | 1 0 1 0

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	2	2	2	3

The current accumulator is rotated one bit to the right. The least-significant bit replaces the most-significant bit. The accumulator is affected only to the set precision.

RR8 Rotate right 8 bits (1 byte)

$$(Acc_n) \leftarrow (Acc_{n+8})$$

$$(Acc_{MSByte}) \leftarrow (Acc_{LSByte})$$

7 6 5 4 | 3 2 1 0

0 0 0 1 | 1 0 1 1

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	2	2	2

The current accumulator is rotated eight bits to the right. The least-significant byte replaces the most-significant byte. The accumulator is affected only to the set precision.

non-memory one-byte instructions, shift

SL1 Shift left 1 bit

$$(Acc_n) \leftarrow (Acc_{n-1})$$

$$(Acc_{LSBit}) \leftarrow 0$$

7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	0

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	2	2	2

The current accumulator is shifted one bit to the left. ZERO is shifted into the least-significant bit. The accumulator is affected only to the set precision.

SL8 Shift left 8 bits (1 byte)

$$(Acc_n) \leftarrow (Acc_{n-8})$$

$$(Acc_{LSByte}) \leftarrow 0$$

7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	1

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	2	2	2

The current accumulator is shifted eight bits to the left. ZEROs are shifted into the least-significant byte. The accumulator is affected only to the set precision.

RL1 Rotate left 1 bit

$$(Acc_n) \leftarrow (Acc_{n-1})$$

$$(Acc_{LSBit}) \leftarrow (Acc_{MSBit})$$

7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	1

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	2	2	2	3

The current accumulator is rotated one bit to the left. The most significant bit replaces the least-significant bit. The accumulator is affected only to the set precision.

RL8 Rotate left 8 bits (1 byte)

$$(Acc_n) \leftarrow (Acc_{n-8})$$

$$(Acc_{LSByte}) \leftarrow (Acc_{MSByte})$$

7	6	5	4	3	2	1	0
0	0	0	1	1	1	1	1

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	2	2	2

The current accumulator is rotated bits to the left. The most-significant byte replaces the least-significant byte. The accumulator is affected only to the set precision.

Environmental Control Instructions

CST **Change environmental status**

7 6 5 4 | 3 2 1 0

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	1	1	1	1

The environment is changed. The noncurrent environment becomes the current operating environment. Instruction execution begins with the instruction at the location in the new location counter. Accumulator precision is unchanged from the last time in the new environment. The old location counter, accumulator and index register are unaffected.

Switching between environments is always under program control even when initiated by hardware interrupt. The actual change in environmental status is accomplished by executing a change-status (CST) instruction or a load-Q-and-change-status (CSQ) instruction. These are the only ways to transfer the program from one program sequence to the alternate sequence.

In general-purpose use, the transfer is programmed to make full use of two accumulators and two index registers.

When using the I/O interrupt system to switch modes, the CST instruction is programmed into the memory location associated with the desired interrupt. Other interrupts may occur, but they must be processed by programming a BRM (branch and mark) instruction in the associated interrupt cells.

While in environment 2 (noninterruptable), whether for general-purpose use or high-rate I/O servicing, external interrupts are ignored. The CST instruction from environment 1 to environment 2 disables any interrupts. The CST instruction from environment 2 back to environment 1 reenables the interrupt capability.

NON-MEMORY REFERENCE TWO-BYTE INSTRUCTIONS

Register-Operate Instructions

The eight 16-bit registers used in maintaining a foreground and background process may also be used as general-purpose registers.

The registers are treated as separate 16-bit values without regard to accumulator precision. The register status (current or noncurrent) is always relative to the current environment (1 or 2).

Register codes for these instructions are as follows:

Register	Binary Code
Z	0000
A	0001
C	0010
X	0011
P	0100
B	0110
D	0111
Y	0111
Q	1000
F*	1001

*The only permissible instruction in which this register may be used is T (transfer).

T Transfer source register to destination register

(d) ← (s)

7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	0

First Byte

7	6	5	4	3	2	1	0
s				d			

Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	3	3	3	3

The contents of the source register are placed in the destination register. The source register is unaffected.

non-memory reference two-byte instructions, register-operate

C Complement source register and transfer to destination register

(d) $\leftarrow (\text{s})$

7 6 5 4 | 3 2 1 0

0 0 1 0 0 0 0 1

First Byte

7 6 5 4 | 3 2 1 0

s d

Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	3	3	3	3

The contents of the source register are inverted and placed in the destination register. The source register is unaffected unless it is also the destination register.

I Increment source register and transfer to destination register

(d) $\leftarrow (\text{s}) + 1$

7 6 5 4 | 3 2 1 0

0 0 1 0 0 0 0 1 0

First Byte

7 6 5 4 | 3 2 1 0

s d

Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	3	3	3	3

The contents of the source register are incremented by one and placed in the destination register. The source register is unaffected unless it is also the destination register.

D Decrement source register and transfer to destination register

(d) $\leftarrow (\text{s}) - 1$

7 6 5 4 | 3 2 1 0

0 0 1 0 0 0 0 1 1

First Byte

7 6 5 4 | 3 2 1 0

s d

Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	3	3	3	3

The contents of the source register are decremented by one and placed in the destination register. The source register is unaffected unless it is also the destination register.

M AND source register into destination register

(d) $\leftarrow (\text{d}) \cap (\text{s})$

7 6 5 4 | 3 2 1 0

0 0 1 0 0 0 1 0 0

First Byte

7 6 5 4 | 3 2 1 0

s d

Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	3	3	3	3

The logical product of the contents of the source register and the destination register is placed in the destination register. The source register is unaffected.

non-memory reference two-byte instructions, register-operate

- O Inclusive OR source register into destination register**

7	6	5	4	3	2	1	0
0	0	1	0	0	1	0	1

First Byte

7	6	5	4	3	2	1	0
s				d			

Second Byte

$$(d) \leftarrow (d) \cup (s)$$

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	3	3	3	3

The logical sum of the contents of the source register and the destination register is placed in the destination register. The source register is unaffected.

- E Exclusive OR source register into destination register**

7	6	5	4	3	2	1	0
0	0	1	0	0	1	1	0

First Byte

7	6	5	4	3	2	1	0
s				d			

Second Byte

$$(d) \leftarrow (d) \oplus (s)$$

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	3	3	3	3

The logical exclusive OR of the contents of the source register and the destination register is placed in the destination register. The source register is unaffected unless it is also the destination register.

- A Add source register to destination register**

7	6	5	4	3	2	1	0
0	0	1	0	0	1	1	1

First Byte

7	6	5	4	3	2	1	0
s				d			

Second Byte

$$(d) \leftarrow (d) + (s)$$

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	3	3	3	3

The arithmetic sum of the contents of the source register and the destination register is placed in the destination register. The source register is unaffected unless it is also the destination register. The overflow indicator is not set by this instruction.

Non-Memory Reference Two-Byte Instructions, Input/Output

Input/Output Instructions

BTO Byte transfer out

I/O data bus $\leftarrow$ (Acc_{LSByte})

I/O address bus $\leftarrow$ o, a

7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
0 0 1 1 0 0 0 0	o a
First Byte	Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	2	2	2	2

The order code and device address are placed on the I/O address bus. The least-significant eight bits of the current accumulator are placed on the I/O data bus.

BTI Byte transfer in

(Acc_{LSByte}) $\leftarrow$ I/O data bus

I/O address bus $\leftarrow$ o, a

7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
0 0 1 1 0 0 0 1	o a
First Byte	Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	2	2	2	2

The order code and device address are placed on the I/O address bus. A byte of data from the addressed device is placed in the least-significant eight bits of the current accumulator.

SEN Sense and skip

I/O address bus $\leftarrow$ o, a

(P) $\leftarrow$ (P) + 2 + 2 [if sense response true]

7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
0 0 1 1 0 0 1 0	o a
First Byte	Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	3	3	3	3

The order code and device address are placed on the I/O address bus. The next two bytes in sequence are skipped if the sense response line on the I/O bus is true. The device being interrogated uses the order code to condition its response to various inquiries.

non-memory reference two-byte instructions, input/output

FUN Function out

I/O data bus $\leftarrow$ (Acc_{LSByte})

I/O address bus $\leftarrow$ o, a

7 6 5 4 | 3 2 1 0

0	0	1	1	0	0	1	1
---	---	---	---	---	---	---	---

First Byte

7 6 5 | 4 3 2 1 0

o	a
---	---

Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	2	2	2	2

The order code and device address are placed on the I/O address bus. The least-significant eight bits of the current accumulator are placed on the I/O data bus (may be used as a special BTO instruction or may be ignored). The device being commanded interprets the order code and executes the indicated function (e.g., rewind tape).

Assigned Input/Output Instructions

Device address 00 is decoded internally by the 520/i to execute special instructions.

MIN Modify interrupts (0 to 3)

Interrupt enable $n = 1$ [if $(Acc_n) = 1$]

Interrupt enable $n = 0$ [if $(Acc_{n+4}) = 1$]

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
First Byte								Second Byte							

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	2	2	2	2

The interrupts are selectively enabled or disabled by this instruction.

The least significant bits of the accumulator are used to modify the enable/disable status of the four interrupts lines. Binary ONES in bits 0 to 3 of the current accumulator enable interrupts 0 to 3. Binary ONES in bits 4 to 7 of the current accumulator disable interrupts 0 to 3.

If the enable bit in the accumulator is a ZERO and the disable bit is a ZERO, the interrupt status is unaffected. If both enable and disable bits are ONE, the interrupt status is complemented.

SS1 Skip if sense switch 1 is off

$(P) \leftarrow (P) + 2 + 2$ [if sense switch 1 = 0]

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	1	1	0	0	1	0	0	0	1	0	0	0	0	0
First Byte								Second Byte							

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	3	3	3	3

If the front-panel SENSE switch 1 is not set, the next two bytes in sequence are skipped.

SS2 Skip if sense switch 2 is off

$(P) \leftarrow (P) + 2 + 2$ [if sense switch 2 = 0]

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	1	1	0	0	1	0	0	1	0	0	0	0	0	0
First Byte								Second Byte							

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	3	3	3	3

If the front-panel SENSE switch 2 is not set, the next two bytes in sequence are skipped.

non-memory reference two-byte instructions, assigned input/output

SS3 Skip if sense switch 3 if off

$(P) \leftarrow (P) + 2 + 2$ [if sense switch 3 = 0]

7	6	5	4	3	2	1	0
0	0	1	1	0	0	1	0

First Byte

7	6	5	4	3	2	1	0
1	0	0	0	0	0	0	0

Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	3	3	3	3

If the front-panel SENSE switch 3 is not set, the next two bytes in sequence are skipped.

SIE Skip on interruptable environment

$(P) \leftarrow (P) + 2 + 2$ [if machine is in environment 1]

7	6	5	4	3	2	1	0
0	0	1	1	0	0	1	0

First Byte

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0

Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	3	3	3	3

If the machine is in the interruptable environment (environment 1), the next two bytes in sequence are skipped.

SSH Skip on halt-mode power failure

$(P) \leftarrow (P) + 2 + 2$ [if machine is in halt-mode during power-fail interrupt]

7	6	5	4	3	2	1	0
0	0	1	1	0	0	1	1

First Byte

7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	0

Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	3	3	3	3

If the machine is in the halt-mode and a power-fail interrupt (optional) is detected, the next two bytes in sequence are skipped.

CIS Copy interrupt status

$(Acc_0 \text{ to } Acc_7) \leftarrow \text{interrupts 3 to 10}$

7	6	5	4	3	2	1	0
0	0	1	1	0	0	0	1

First Byte

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0

Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	2	2	2	2

The eight interrupt lines that share priority interrupt 3 are placed in the least-significant bits of the current accumulator.

non-memory reference two-byte instructions, assigned input/output

CIM Copy interrupt mask

(Acc₀ to Acc₃) ← Interrupt Mask

7 6 5 4 | 3 2 1 0

0 0 1 1 0 0 0 1

First Byte

7 6 5 4 | 3 2 1 0

0 0 1 0 0 0 0 0

Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	2	2	2	2

The four interrupt mask lines are placed in the least-significant bits of the current accumulator.

DBE Interrupt block disable

7 6 5 4 | 3 2 1 0

0 0 1 1 0 0 1 1

First Byte

7 6 5 4 | 3 2 1 0

0 0 0 0 0 0 0 0

Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	2	2	2	2

The interrupt system is disabled as a block.

TBE Interrupt block enable

7 6 5 4 | 3 2 1 0

0 0 1 1 0 0 1 1

First Byte

7 6 5 4 | 3 2 1 0

0 0 1 0 0 0 0 0

Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	2	2	2	2

The interrupt system is enabled as a block.

Non-Memory Reference Two-Byte Instructions, Teletype Controller Reserved

Teletype Controller Reserved Instructions

Device address 01 is reserved for the teletype controller.

FUN Reset teletype controller

I/O data bus $\leftarrow$ (Acc_{LSByte})

7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
0 0 1 1 0 0 1 1	0 0 1 0 0 0 0 1
First Byte	Second Byte

Operation Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	2	2	2	2

The instruction places the teletype controller in the output buffer ready condition, halts the paper tape reader of the teletype and disables both teletype controller interrupt masks.

FUN Start teletype paper tape reader

I/O data bus $\leftarrow$ (Acc_{LSByte})

7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
0 0 1 1 0 0 1 1	0 1 0 0 0 0 0 1
First Byte	Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	2	2	2	2

Continuous paper tape motion is initiated in the paper tape reader of the teletype.

FUN Enable teletype read interrupt

I/O data bus $\leftarrow$ (Acc_{LSByte})

7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
0 0 1 1 0 0 1 1	0 1 1 0 0 0 0 1
First Byte	Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	2	2	2	2

The teletype controller read interrupt is enabled.

FUN Enable teletype write interrupt

I/O data bus $\leftarrow$ (Acc_{LSByte})

7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
0 0 1 1 0 0 1 1	0 0 0 0 0 0 0 1
First Byte	Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	2	2	2	2

The teletype controller write interrupt is enabled.

non-memory reference two-byte instructions, teletype controller reserved

FUN Read one character

I/O data bus $\leftarrow$ (Acc_{LSByte})

7 6 5 4 | 3 2 1 0

0 0 1 1 0 0 1 1

First Byte

7 6 5 4 | 3 2 1 0

1 1 0 0 0 0 0 1

Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	2	2	2	2

The paper tape reader of the teletype is advanced one character and stops on the following character.

BTI Transfer teletype controller buffer to the C register

(C_{LSByte}) $\leftarrow$ (Teletype Controller Buffer)

7 6 5 4 | 3 2 1 0

0 0 1 1 0 0 0 1

First Byte

7 6 5 4 | 3 2 1 0

0 0 0 0 0 0 0 1

Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	2	2	2	2

The contents of the teletype controller buffer are transferred to the least-significant bits of the current accumulator.

BTO Transfer C register to the teletype controller buffer

(Teletype Controller Buffer) $\leftarrow$ (C_{LSByte})

7 6 5 4 | 3 2 1 0

0 0 1 1 0 0 0 0

First Byte

7 6 5 4 | 3 2 1 0

0 0 0 0 0 0 0 1

Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	2	2	2	2

The contents of the least-significant bits of the current accumulator are transferred to the teletype controller buffer.

SEN Sense teletype output buffer ready

(P) $\leftarrow$ (P) + 2 + 2 [if WRDY = 1]

7 6 5 4 | 3 2 1 0

0 0 1 1 0 0 1 0

First Byte

7 6 5 4 | 3 2 1 0

0 0 1 0 0 0 0 1

Second Byte

Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	3	3	3	3

If the teletype output-buffer-ready signal is true, the next two bytes in sequence are skipped.

non-memory reference two-byte instructions, teletype controller reserved

SEN Sense teletype input buffer ready

7 6 5 4 | 3 2 1 0

7 6 5 4 | 3 2 1 0

$(P) \leftarrow (P) + 2 + 2$ [if RRDY = 1]

0	0	1	1	0	0	1	0
---	---	---	---	---	---	---	---

0	1	0	0	0	0	0	1
---	---	---	---	---	---	---	---

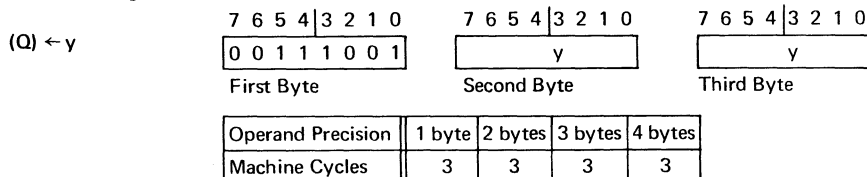
First Byte

Second Byte

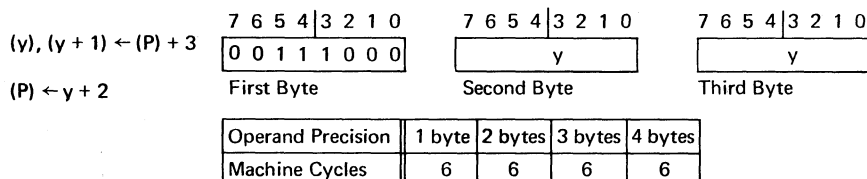
Operand Precision	1 byte	2 bytes	3 bytes	4 bytes
Machine Cycles	3	3	3	3

If the teletype input-buffer-ready signal is true, the next two bytes in sequence are skipped.

NON-MEMORY-REFERENCE, THREE-BYTE INSTRUCTIONS

CSQ Change environmental status and load the noncurrent program counter

The environment is changed. The noncurrent environment becomes the current operating environment. The last two bytes of the CSQ instruction are loaded into the location counter of the environment just becoming current.

BRM Branch and mark

The program counter, after being incremented to the next instruction location, is placed in memory locations y and $y + 1$ specified by the BRM instruction. The address $y + 2$ is placed in the program counter.

ORG Set location counter

This instruction alters the contents of the location counter to the value specified in the variable field. A symbol in the variable field must have been previously defined. A negative or undefined quantity in the variable field is ignored and causes no change to the program counter. A symbol in the label field is assigned the value in the variable field.

BSS Block storage specification

This instruction reserves a block of memory locations (without affecting the contents) and assigns a name to the block. The size of the block is defined by the quantity in the variable field. Memory locations in the block are sequential. The label refers to the location with the lowest address. Successive locations are referred to as label +1, label+2, etc.

LIT Begin literal blocks

This instruction reserves a block of memory locations for all literals that have occurred since the previous LIT instruction or since the start of the program. The value in the variable field specifies the lowest address of the block. A symbol in the variable field must have been previously defined. An undefined symbol, invalid quantity or a blank in the variable field causes literals to be assigned locations beginning with the current location counter.

If a LIT instruction is not included in the source program, all literals in the program are assigned locations starting with the first location after the last address in the program.

IOR Begin indirect-pointer block

This instruction reserves a block of memory locations for indirect addresses in the object program. The value in the variable field specifies the lowest address of the block and must not be larger than \$3FA if the block is to accommodate the maximum number (3) of indirect addresses within the lowest (zero) sector of the object program.

The IOR instruction must appear in the source program before any indirect addresses are generated.

If the programmer includes an IOR instruction in the source program, the assembly program generates an indirect memory reference whenever an addressing violation cannot be resolved. The assembly program then inserts a reference to the indirect-pointer block in the object instruction along with an indirect-through-the-zero-sector addressing code. The actual indirect memory address generated by the assembly program is stored in the indirect-pointer block at the reference location inserted in the object instruction. The assembly program prints an I on the assembly listing immediately following the operation-code appendage of the source instruction for which the indirect address is generated.

SET Set symbol value

This instruction changes the address value of the symbol in the label field to the value in the variable field. A symbol in the variable field must have been previously defined.

EQU Equate symbols

This instruction assigns the symbol in the label field an address value equal to the value in the variable field. A symbol in the variable field must have been previously defined.

MOR Halt for more input

This instruction provides the programmer control of the program input mode during assembly pass 1. When the assembly program encounters a MOR instruction, the computer halts at location \$61B.

If further input is to be made from the keyboard, the programmer must press the RUN switch. The assembly program then issues a carriage return followed by two line feeds to the teletype and the programmer may begin typing source statements.

If further input is to be made from punched tape, the programmer must load the source program tape to some part of the blank leader preceding the first statement, manually enter the value \$C1 (binary 1100 0001) into the C register of the 520/i and press the RUN switch.

END End assembly

This instruction terminates assembly of the program and must be the last statement of a source program. As a result of this instruction, the program counter is set to the value in the variable field. An undefined symbol, invalid quantity or a blank in the variable field causes the program counter to be reset to zero.

DA1,DA2,DA3,DA4 Define constant

This instruction defines constant data in storage. The precision of each defined data word, in bytes, is the last character of the instruction mnemonic. For example, DA3 specifies data to a precision of three bytes (24 bits).

This instruction may specify one or a series of constants. A data constant is any valid octal, decimal or hexadecimal integer. An alpha constant is any series of alphanumeric characters enclosed between two prime (') characters. (A double prime is interpreted as a single prime when it is part of an alpha constant.) An address constant is any valid address symbol. Multiple constants in a data-definition statement must be separated by commas. More than one type of constant may be included in a data definition statement.

A symbol in the label field is given the address of the leftmost byte of the variable field. Each constant value in the variable field, except an alpha constant, is allotted the number of bytes of storage specified by the last character of the instruction mnemonic. The constants are stored in the rightmost (highest-numbered address) bytes of the allotted locations. Alpha constants are allotted one byte of storage for each character. If the total storage block allotted for a data-definition instruction is not an even multiple of the specified precision, bytes of zeros are stored to fill the difference. The specified number of bytes filled with zeros are allotted for invalid data constants and side-by-side commas in the variable field, and for a blank or carriage return following a comma at the end of a source statement.

Mnemonic	Hexadecimal Code	Definition
A	27--	Add source register to destination register
A	1	Current accumulator most-significant 16 bits
AND	CO--	AND memory to the current accumulator, direct
AND	DO--	AND memory to the current accumulator, direct, current sector
AND	D4--	AND memory to the current accumulator, indirect, current sector
AND	D8--	AND memory to the current accumulator, indexed
AND	DC--	AND memory to the current accumulator, indirect, zero sector
B	5	Noncurrent accumulator most-significant 16 bits
BRM	38-----	Branch and mark (store) current program counter
BRU	EO---	Branch unconditionally, direct
BRU	FO---	Branch unconditionally, direct, current sector
BRU	F4--	Branch unconditionally, indirect, current sector
BRU	F8--	Branch unconditionally, indexed
BRU	FC--	Branch unconditionally, indirect, zero sector
BTI 02	3102	Read one tape character into least significant byte of current accumulator
BTI	31--	Byte transfer into current accumulator
BTO 01	3001	Transfer least-significant byte of current accumulator to the teletype controller input buffer

alphabetical index of instructions

Mnemonic	Hexadecimal Code	Definition
BTI 01	3101	Transfer teletype controller buffer to the least-significant byte of the current accumulator
BTO 02	3002	Punch least-significant byte of current accumulator
BTO	30--	Byte transfer out of current accumulator
C	21 s,d	Complement source register and transfer to destination register
C	2	Current accumulator least-significant 16 bits
CIA	0F	Complement and increment current accumulator to the set precision
CIM	3120	Copy interrupt mask into current accumulator
CIS	3100	Copy interrupt status into current accumulator
CLA	0C	Clear current accumulator to the set precision
CMA	0D	Complement current accumulator to the set precision
CSQ	39----	Change environmental status and load noncurrent program counter
CST	01	Change environmental status
D	23 s,d	Decrement source register and transfer to destination register
D	6	Noncurrent accumulator least-significant 16 bits
DBE	3300	Interrupt system disable
E	26 s,d	Exclusive OR source register into destination register
F	9	Peripheral adapter input/output register

Mnemonic	Hexadecimal Code	Definition
FUN 01	3301	Enable teletype controller write interrupt
FUN 22	3322	Start punched-tape reader
FUN 31	3321	Reset teletype controller
FUN	3	Transfer function out and transfer a byte out of the current accumulator
FUN 41	3341	Start teletype punched-tape reader
FUN 42	3342	Enable punched-tape interrupt
FUN 61	3361	Enable teletype controller read interrupt
FUN A2	33A2	Stop punched-tape reader
FUN C1	33C1	Read one character on the teletype punched-tape reader
FUN C2	33C2	Disable punched-tape interrupt
HLT	00	Halt with program counter at next location
I	22 s,d	Interrupt source register and transfer to destination register
IAR	OE	Increment current accumulator to the set precision
IBE	3320	Interrupt system enable
IXO	17	Increment current index register by the set precision
LOD	40--	Load the current accumulator from memory, direct addressing mode

alphabetical index of instructions

Mnemonic	Hexadecimal Code	Definition
LOD	50--	Load the current accumulator from memory, direct, current sector
LOD	54--	Load the current accumulator from memory, indirect, current sector
LOD	58--	Load the current accumulator from memory, indexed
LOD	5C--	Load the current accumulator from memory, indirect, zero sector
M	24 s d	AND source register into destination register
MIN	3000	Modify interrupts according to current accumulator
NOP	16	No operation
O	25 s d	OR source register into destination register
Q	8	Noncurrent program counter
P	4	Current program counter
RL1	1E	Rotate current accumulator left one bit to the set precision
RL8	1F	Rotate current accumulator left eight bits to the set precision
ROF	15	Reset current overflow indicator
RR1	1A	Rotate current accumulator right one bit to the set precision
RR8	1B	Rotate current accumulator right eight bits to the set precision

Mnemonic	Hexadecimal Code	Definition
SAN	06	Skip if current accumulator is negative
SANN	07	Skip if current accumulator is positive
SAZ	08	Skip if current accumulator is zero
SAZN	09	Skip if current accumulator is not zero
SEN 02	3202	Sense for punched-tape transfer readiness and skip if ready
SEN 21	3221	Sense for teletype controller output buffer readiness and skip if ready
SEN	32	Sense device and skip if sense true
SEN 41	3241	Sense for teletype controller input buffer readiness and skip if ready
SIE	3200	Skip if in interruptable environment
SLI	1C	Shift current accumulator left one bit to the set precision
SL8	1D	Shift current accumulator left eight bits to the set precision
SOD	04	Skip if current accumulator is odd
SODN	05	Skip if current accumulator is not odd
SOF	14	Set current overflow indicator
SOV	02	Skip on current accumulator overflow and reset overflow indicator
SOVN	03	Skip on no current accumulator overflow and reset overflow indicator

alphabetical index of instructions

Mnemonic	Hexadecimal Code	Definition
SP1	10	Set current operand precision to 1 byte
SP2	11	Set current operand precision to 2 bytes
SP3	12	Set current precision to 3 bytes
SR4	18	Shift current accumulator right one bit to the set precision
SR8	19	Shift current accumulator right eight bits to the set precision
SS2	3240	Skip if SENSE switch 2 is off
SS3	3280	Skip if SENSE switch 3 is off
SSH	3260	Skip on power-fail interrupt in halt mode
SS1	3220	Skip if SENSE switch 1 is off
ST0	60-	Store the current accumulator in memory, direct
ST0	70-	Store the current accumulator in memory, direct, current sector
ST0	74-	Store the current accumulator in memory, indirect, current sector
ST0	78-	Store the current accumulator in memory, indexed
ST0	7C-	Store the current accumulator in memory, indirect, zero sector
SXZ	0A	Skip if current index register is zero
SXZN	0B	Skip if current index register is not zero

Mnemonic	Hexadecimal Code	Definition
SUB	A0--	Subtract memory from the current accumulator, direct
SUB	BC--	Subtract memory from the current accumulator
SUB	BO--	Subtract memory from the current accumulator, direct, current sector
SUB	B4--	Subtract memory from the current accumulator, indirect, current sector
SUB	B8--	Subtract memory from the current accumulator, indexed
SUM	80--	Add memory to the current accumulator, direct
SUM	90--	Add memory to the current accumulator, direct, current sector
SUM	94--	Add memory to the current accumulator, indirect, current sector
SUM	98--	Add memory to the current accumulator, indexed
SUM	9C	Add memory to the current accumulator, indirect, zero sector
T	20 s,d	Transfer source register to destination register
X	3--	Current index register
Y		Noncurrent index register
Z		Nonexistent zeros register

Numerical Index of Instructions

Hexadecimal Code	Mnemonic	Definition
00	HLT	Halt with program counter at next location
0	Z	Nonexistent zeros register
01	CST	Change environmental status
1	A	Current accumulator most-significant 16 bits
02	SOV	Skip on current accumulator overflow and reset overflow indicator
2	C	Current accumulator least-significant 16 bits
03	SOVN	Skip on no current accumulator overflow and reset overflow indicator
3	X	Current index register
04	SOD	Skip if current accumulator is odd
4	P	Current program counter
05	SODN	Skip if current accumulator is not odd
5	B	Noncurrent accumulator most-significant 16 bits
06	SAN	Skip if current accumulator is negative
6	D	Noncurrent accumulator least-significant 16 bits
07	SANN	Skip if current accumulator is positive
7	Y	Noncurrent index register
08	SAZ	Skip if current accumulator is zero
8	Q	Noncurrent program counter

Hexadecimal Code	Mnemonic	Definition
09	SAZN	Skip if current accumulator is not zero
9	F	Peripheral adapter input/output register
0A	SXZ	Skip if current index register is zero
0B	SXZN	Skip if current index register is not zero
0C	CLA	Clear current accumulator to the set precision
0D	CMA	Complement current accumulator to the set precision
0E	IAR	Increment current accumulator to the set precision
0F	CIA	Complement and increment current accumulator to the set precision
10	SP1	Set current operand precision to 1 byte
11	SP2	Set current operand precision to 2 bytes
12	SP3	Set current operand precision to 3 bytes
14	SOF	Set current overflow indicator
15	ROF	Reset current overflow indicator
16	NOP	No operation
17	IXO	Increment current index register by the set precision
18	SR1	Shift current accumulator right one bit to the set precision
19	SR8	Shift current accumulator right eight bits to the set precision

numerical index of instructions

Hexadecimal Code	Mnemonic	Definition
1A	RR1	Rotate current accumulator right one bit to the set precision
1B	RR8	Rotate current accumulator right eight bits to the set precision
1C	SL1	Shift current accumulator left one bit to the set precision
1D	SL8	Shift current accumulator left eight bits to the set precision
1E	RL1	Rotate current accumulator left one bit to the set precision
1F	RL8	Rotate current accumulator left eight bits to the set precision
20 s d	T	Transfer source register to destination register
21 s d	C	Complement source register and transfer to destination register
22 s d	I	Increment source register and transfer to destination register
23 s d	D	Decrement source register and transfer to destination register
24 s d	M	AND source register into destination register
25 s d	O	OR source register into destination register
26 s d	E	Exclusive OR source register into destination register
27 s d	A	Add source register to destination register

Hexadecimal Code	Mnemonic	Definition
30—	BTO	Byte transfer out of current accumulator
31—	BTI	Byte transfer into current accumulator
33—	SEN	Sense device and ship is sense true
34—	FUN	Transfer function out and transfer a byte out of the current accumulator
38—	BRM	Branch and mark (store) current program counter
39—	CSQ	Change environmental status and load noncurrent program counter
40—	LOD	Load the current accumulator from memory, direct addressing mode
50—	LOD	Load the current accumulator from memory, direct, current sector
54—	LOD	Load the current accumulator from memory, indirect, current sector
58—	LOD	Load the current accumulator from memory, indexed
5C—	LOD	Load the current accumulator from memory, indirect, zero sector
60—	STO	Store the current accumulator in memory, direct
70—	STO	Store the current accumulator in memory, direct, current sector
74—	STO	Store the current accumulator in memory, indirect, current sector
78—	STO	Store the current accumulator in memory, indexed

numerical index of instructions

Hexadecimal Code	Mnemonic	Definition
7C	STO	Store the current accumulator in memory, indirect, zero sector
80—	SUM	Add memory to the current accumulator, direct
90—	SUM	Add memory to the current accumulator, direct, current sector
94—	SUM	Add memory to the current accumulator, indirect, current sector
98—	SUM	Add memory to the current accumulator, indexed
9C	SUM	Add memory to the current accumulator, indirect, zero sector
A0—	SUB	Subtract memory from the current accumulator, direct
B0—	SUB	Subtract memory from the current accumulator, direct, current sector
B4—	SUB	Subtract memory from the current accumulator, indirect, current sector
B8—	SUB	Subtract memory from the current accumulator, indexed
BC—	SUB	Subtract memory from the current accumulator
CO—	AND	AND memory to the current accumulator, direct
DO—	AND	AND memory to the current accumulator, direct, current sector
D4—	AND	AND memory to the current accumulator, indirect, current sector

Hexadecimal Code	Mnemonic	Definition
D8—	AND	AND memory to the current accumulator, indexed
DC—	AND	AND memory to the current accumulator, indirect, zero sector
EO—	BRU	Branch unconditionally, direct
FO—	BRU	Branch unconditionally, direct, current sector
F4—	BRU	Branch unconditionally, indirect, current sector
F8—	BRU	Branch unconditionally, indexed
FC—	BRU	Branch unconditionally, indirect, zero sector
3000	MIN	Modify interrupts according to current accumulator
3001	BTO 01	Transfer least-significant byte of current accumulator to the teletype controller input buffer
3002	BTO 02	Punch least-significant byte of current accumulator
3100	CIS	Copy interrupt status into current accumulator
3101	BTI 01	Transfer teletype controller buffer to the least-significant byte of the current accumulator
3102	BTI 02	Read one tape character into least significant byte of current accumulator
3120	CIM	Copy interrupt mask into current accumulator
3200	SIE	Skip if in interruptable environment

numerical index of instructions

Hexadecimal Code	Mnemonic	Definition
3202	SEN 02	Sense for punched-tape transfer readiness and skip if ready
3220	SS1	Skip if SENSE switch 1 is off
3221	SEN 21	Sense for teletype controller output buffer readiness and skip if ready
3240	SS2	Skip if SENSE switch 2 is off
3241	SEN 41	Sense for teletype controller input buffer readiness and skip if ready
3260	SSH	Skip on power-fail interrupt in halt mode
3280	SS3	Skip if SENSE switch 3 is off
3300	IBD	Interrupt system disable
3301	FUN 01	Enable teletype controller write interrupt
3220	IBE	Interrupt system enable
3321	FUN 31	Reset teletype controller
3322	FUN 22	Start punched-tape reader
3341	FUN 41	Start teletype punch-tape reader
3342	FUN 42	Enable punched-tape interrupt
3361	FUN 61	Enable teletype controller read interrupt
33A2	FUN A2	Stop punched-tape reader
33C1	FUN C1	Read one character on the teletype punched-tape reader
33C2	FUN C2	Disable punched-tape interrupt

(Please detach card here)

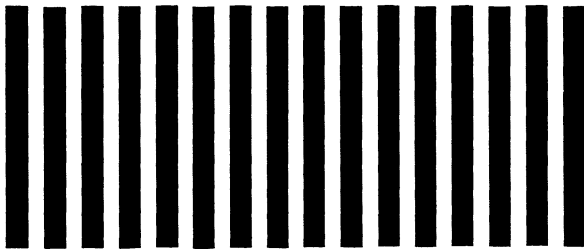


FIRST CLASS
PERMIT NO. 323
NEWPORT BEACH
CALIFORNIA

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN UNITED STATES

POSTAGE WILL BE PAID BY--

varian data machines
2722 Michelson Drive
Irvine, California 92664





I AM INTERESTED IN:

- ☐ **Varian 520/i Computer**
☐ **Varian 520/DC Data Communication System**
☐ **Varian 620/i Computer**
☐ **Varian R 620/i Ruggedized Computer**
☐ **VersaSTORE Core Memory Systems**

Application:

- ☐ Batch processing
☐ Analytical instrumentation data
☐ Telemetry or aerospace test data
☐ Small computer used as a local satellite to a larger computer for control of I/O peripherals
☐ Small computer used as a remote satellite to a larger computer for control of I/O peripherals
☐ Numerical control of machine tools or drafting machines
☐ Control of chemical, petroleum, or power generation processes
☐ Communication switching, data concentration, etc.
☐ Production line testing
☐ Other _____

(PLEASE SPECIFY)

- ☐ **Please have a sales engineer call me**
☐ **Please add my name to your mailing list**

Mail to:

NAME _____

TITLE _____

COMPANY _____

DEPT.
CODE

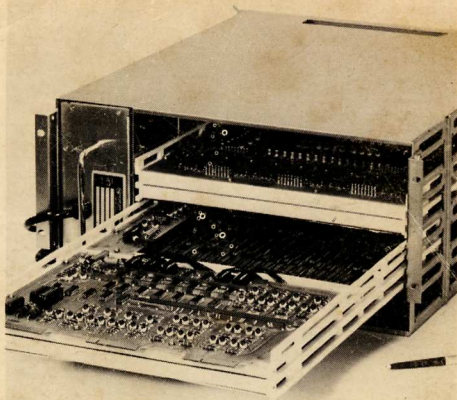
ADDRESS _____

CITY _____ PHONE _____

STATE _____ ZIP # _____

For VDM Use Only

\$3.00



varian **520/i** computer handbook



varian data machines
2722 Michelson Drive
Irvine, California 92664

